Rational
the e-development company

UML para Factorías

Capítulo 11: Diseño de Arquitectura

Objetivos: Diseño de Arquitectura

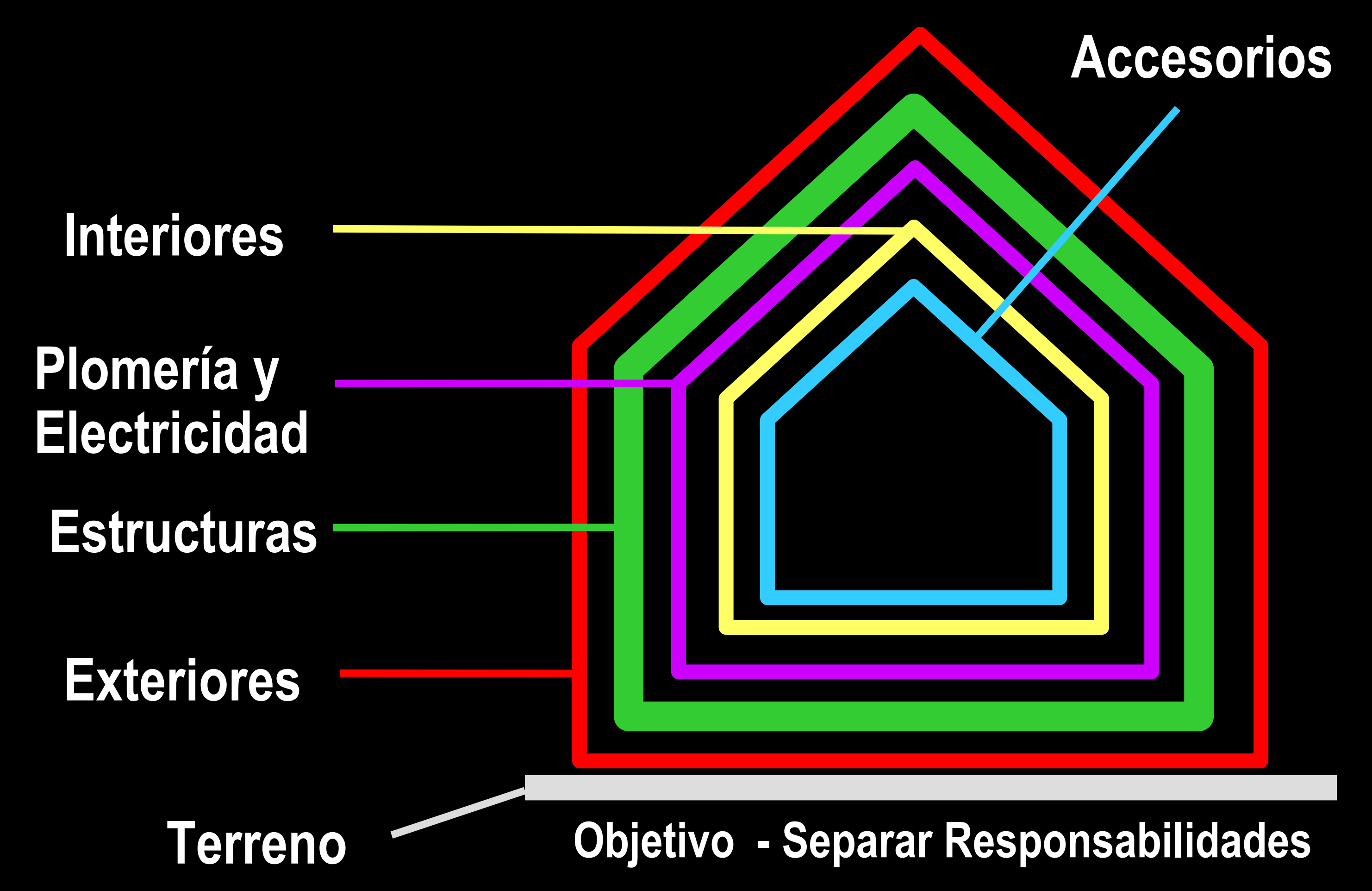
- **Al final de este capitulo, usted podrá:**
  - Entender que es una Arquitectura de Software
  - Definir las características deseables para tener una buena Arquitectura de Software
  - Definir que es un Modelo o Estilo de Arquitectura de Software y que elementos lo definen
  - Conocer las vistas de arquitectura "4+1"
  - Explicar el propósito de los diagramas de componentes, procesos y producción (Deployment)
  - Explicar como se hace el Diseño de Arquitectura de Software
  - Explicar quien hace el Diseño de Arquitectura de Software
  - Explicar como se documenta la Arquitectura de Software



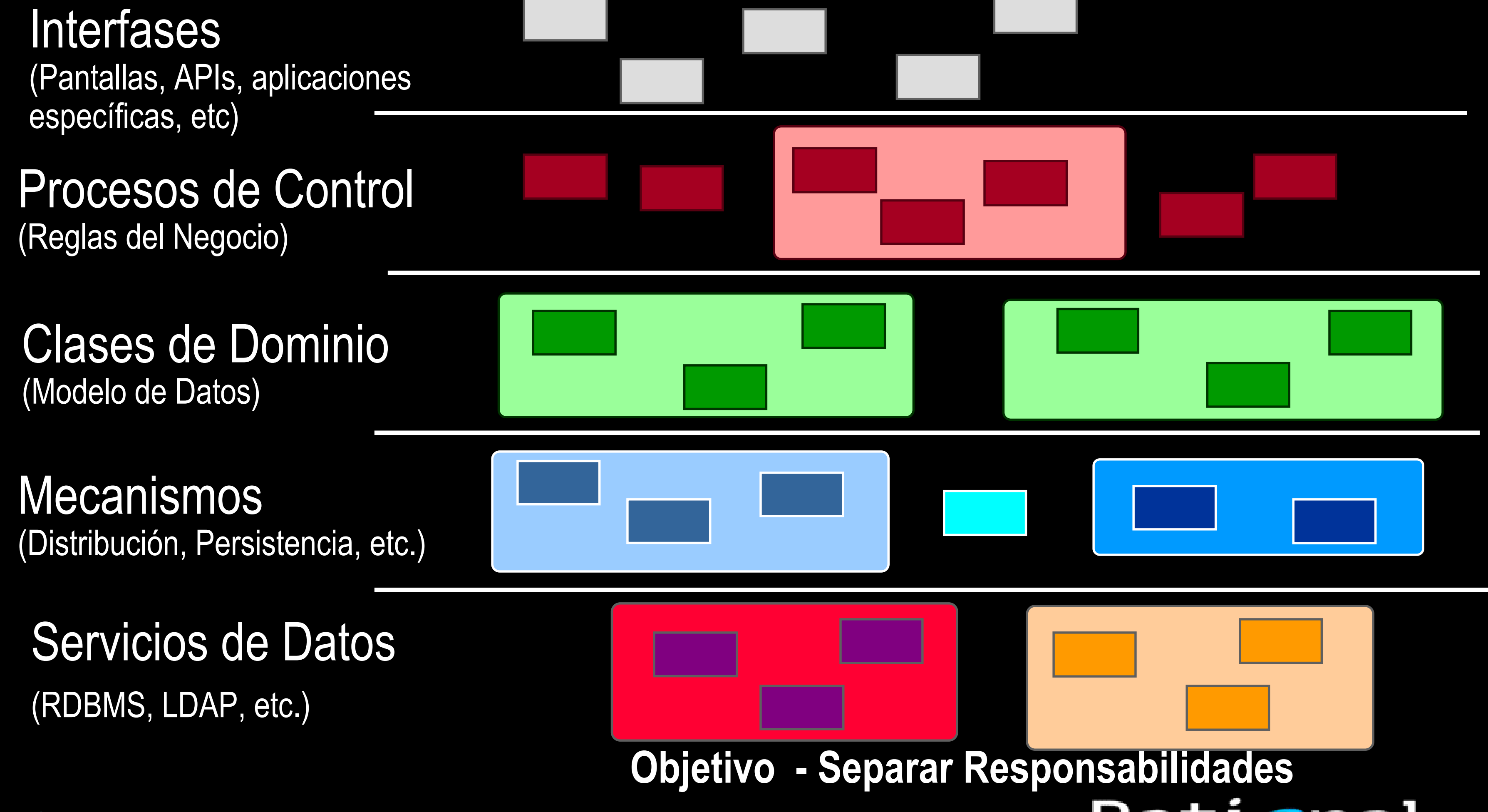
La Arquitectura Decide la Organización del Software

- **La Arquitectura de Software incluye todas la decisiones críticas acerca de la organización del software:**
  - Realiza la seleccion de los elementos estructurales (y sus interfaces) que componen un sistema de software; define la estructura y los componentes.
  - Establece las reglas de colaboracion entre estos elementos; define el funcionamiento.
  - Agrupa estos elementos con estructura y/o funcionalidad similar en subsistemas mas grandes.
  - Establece los estandares de tecnologia, estilo, etc.

Organización de los Elementos que se Utilizan en Arquitectura Civil para Construír una Casa



Organizacion de los Elementos que se Utilizan en la Arquitectura de Software para Construir un Sistema



Objetivo - Separar Responsabilidades

¿Qué es un Componente de Software?

- **Es un elemento de un sistema de software que cumple con los siguientes características:**
  - NO es trivial, es casi independiente y es reemplazable o sustituible
  - Tiene dos partes: un **interfase** y una **implementación** de ese interfase
  - Cumple una funcionalidad especifica dentro del contexto del sistema y a esta funcionalidad se accede usando el interfase de componente
- **Puede verse desde varias perspectivas:**
  - **Componentes de Desarrollo**: son los elementos que se modelan y construyen durante el desarrollo (clases, paquetes, código fuente)
  - **Componentes de Runtime:** representan y agrupan componentes de desarrollo para su ejecución (programas ejecutables y elementos compilados, scripts y elementos interpretables)
  - **Componentes de Negocios**: agrupan componentes de runtime en bloques funcionales (subsistemas, paquetes, módulos, parches) y también representan a los componentes empaquetados o paquetes de uso comercial (COTS, commercial off-the-shelf software)

¿Cómo se Logra una Buena Arquitectura de Software?

Una buena arquitectura se debe estructurar en capas que representan abstracciones bien definidas y con niveles progresivos de dependencia con respecto al sistema que se esta construyendo

- Cada capa se **construye sobre componentes controlados** y bien definidos a un nivel más bajo de abstracción.
- Cada capa tiene una **interfase controlada y bien definida** que establece su funcionalidad; esta interfase esta definida en función de las interfaces de los componentes contenidos en la capa.
- Debe existir una **separación clara entre la interfase y la implementación** de cada capa; los componentes contenidos en la capa pueden modificarse pero su interfase debe respetarse.
- La capas se deben organizar de manera que las mas genéricas (menos dependientes) estén en los niveles bajos y las mas especificas (más dependientes) estén en los niveles mas altos.



Características de las Buenas Arquitecturas de Software

- Las buenas arquitecturas se estructuran en capas con niveles progresivos de abstraccion y dependencia con respecto al sistema que se esta construyendo.
  - Esto promueve la separación de responsabilidades en el equipo de desarrollo, minimiza la dependencia de las plataformas de hardware y estandariza los modelos estructurales
  - Ejemplo: el modelo de 3 capas basado en el patrón de diseño MVC (Model-View-Controller)

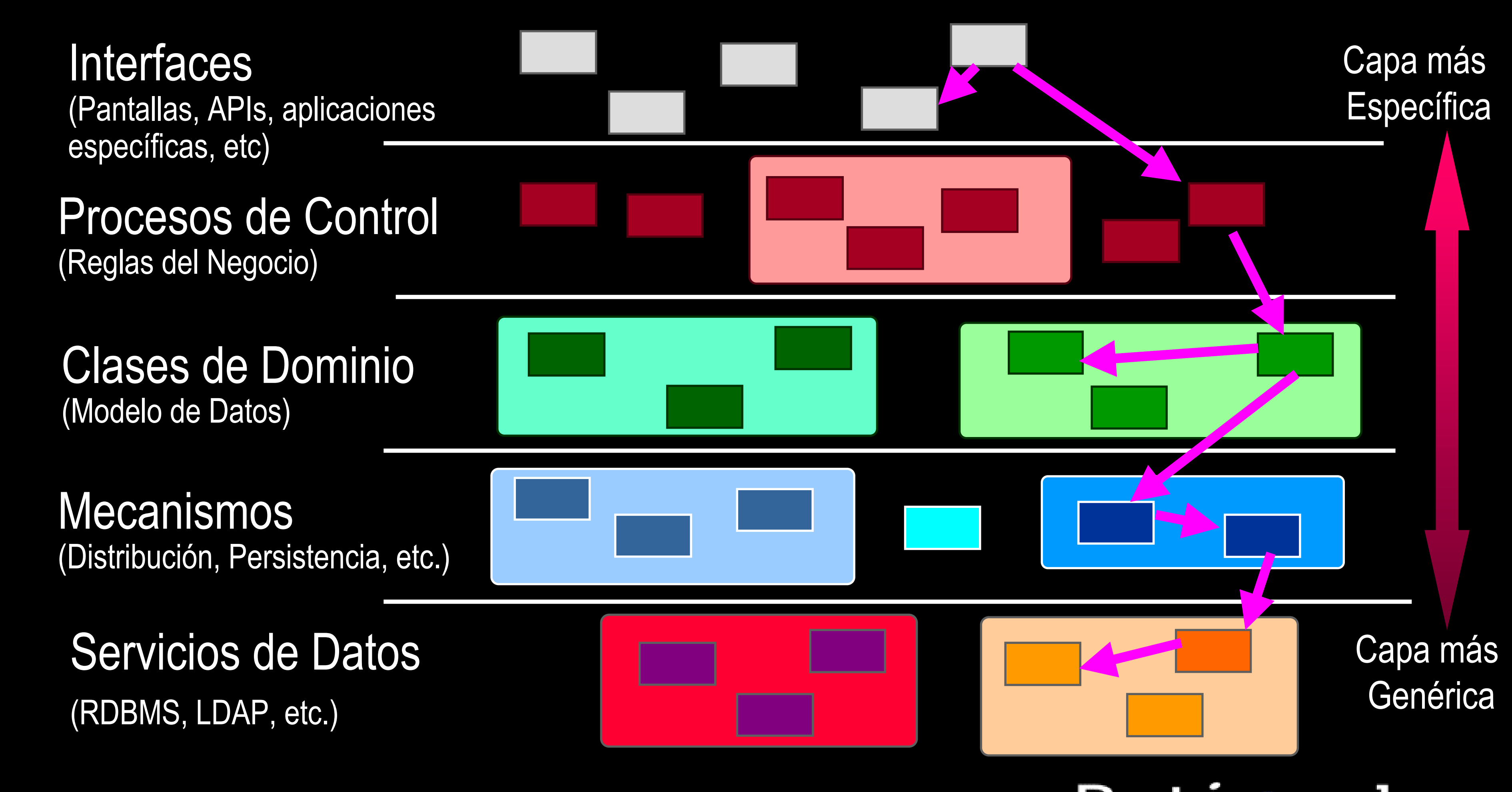
Características de las Buenas Arquitecturas de Software

- En cada capa, se deben utilizar y reutilizar componentes, e infraestructuras de componentes, que han sido estandarizados por la industria
  - Se garantiza la soportabilidad (evolución y mantenimiento), robustez y flexibilidad de los sistemas que se construyen, al usar componentes que separan su interface (que se puede hacer con ellos) de su implementación (como lo hacen).
  - Ejemplo: RDBMS, J2EE, DCOM, .NET, CORBA, ...

Las buenas arquitecturas están basadas en COMPONENTES distribuidos en MULTIPLES CAPAS.



Organizacion de los Capas y Componentes en una Buena Arquitectura de Software



Una Arquitectura Requiere Múltiples Vistas

- **Para describir completamente una arquitectura, se necesitan cuatro vistas:**
  - **La vista lógica que proporciona una imagen estática de las clases primarias y sus relaciones**
  - **La vista de componentes que muestra como el código se organiza en paquetes y componentes según perspectivas de desarrollo y producción**
  - **La vista de procesos muestra como los componentes representan y se distribuyen en procesos**
  - **La vista de producción muestra la distribución de procesos en los nodos (procesadores & dispositivos) y enlaces en el ambiente operacional**
- **Finalmente una vista de casos de uso que sirve de base para las otras cuatro vistas y que explica como funciona el sistema que se esta describiendo en la arquitectura.**

El Modelo de “4+1 Vistas”

Vista Lógica

Funcionalidad

Analistas, Desarrolladores

Vista de Componentes

Manejo de Software,
Reutilización,
Portabilidad

Desarrolladores,
Ingenieros de Software

Vista de Casos de Uso

Entendimiento
Usabilidad

Usuarios Finales,
Analistas

Vista de Procesos

Rendimiento,
Disponibilidad
Tolerancia a Fallas

Integradores de Sistemas

Vista de Producción

Escalabilidad,
Instalación
Configuración

Ingenieros de Sistemas

Vista de Casos de Uso

- **La vista de casos de uso es un subset del modelo de casos de uso**
- **Incluye los casos de uso y escenarios que son criticos para la arquitectura e impulsan el diseño de la misma**
  - Con funcionalidad que tiene dependencias de tecnología o que identifica elementos con riesgo técnico
  - Que tienen dependencias a requerimientos no-funcionales
  - Que identifican y utilizan interfaces críticas
  - Que muestran oportunidades de reutilización
- **Estructura e integra las vistas lógica, de procesos, de desarrollo y de producción de la arquitectura**

Vista Lógica

- La vista lógica es un subset del Modelo de Diseño
- Provee una base para entender la estructura y organización del diseño de un sistema
- Incluye las realizaciones de los casos de uso que están incluidos en la vista de casos de uso
- Se captura en diagramas de clase que contienen los paquetes, clases y relaciones que representan las abstracciones usadas en las realizaciones de los CU y que en conjunto muestran una imagen estática del sistema que se esta desarrollando

Vista de Componentes

- **La vista de componentes muestra la organización real de los componentes de software según el modelo de arquitectura del sistema que se está desarrollando.**
- **Debe incluir dos o mas diagramas de componentes que cumplan con los siguientes propósitos:**
  - Mostrar las relaciones de dependencia entre componentes, según el enfoque de los mismos
  - Organizar los componentes en paquetes que representen una jerarquía de capas

¿Qué es un Componente de Software?

- Es un elemento de un sistema de software que cumple con los siguientes características:
  - NO es trivial, es casi independiente y es reemplazable o sustituible
  - Tiene dos partes: un **interfase** y una **implementación** de ese interfase
  - Cumple una funcionalidad especifica dentro del contexto del sistema y a esta funcionalidad se accede usando el interfase de componente

Enfoque de los Componentes

- **Componentes de Desarrollo** - son los elementos que se modelan y construyen durante el desarrollo, e incluyen:
  - elementos de diseño (clases y paquetes)
  - los archivos con código fuente que representan fisicamente los elementos de diseño

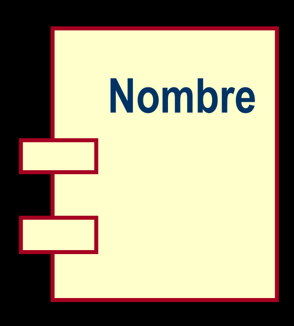
- **Componentes de Runtime** - a estos se les asignan (individualmente o en grupo) los componentes de desarrollo para su ejecución:
  - programas ejecutables y elementos compilados (librerías, archivos de deployment como jars, wars, etc.)
  - elementos interpretables (scripts, archivos con tags, etc.)

Enfoque de los Componentes

- **Componentes de Negocios** - agrupan componentes de runtime en bloques funcionales y tambien representan los componentes empaquetados o paquetes de uso comercial:
  - **subsistemas, paquetes, módulos, parches, COTS**

Representación de Componentes en UML

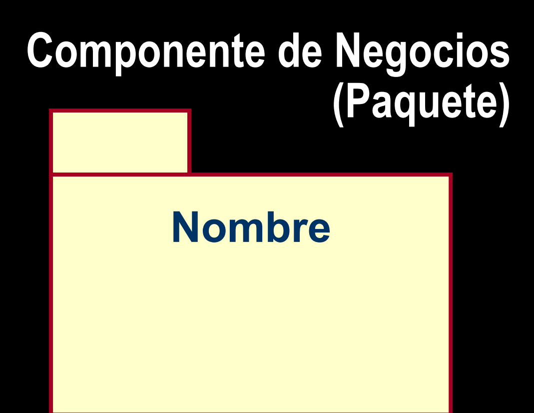
Componente Desarrollo



Componente Runtime

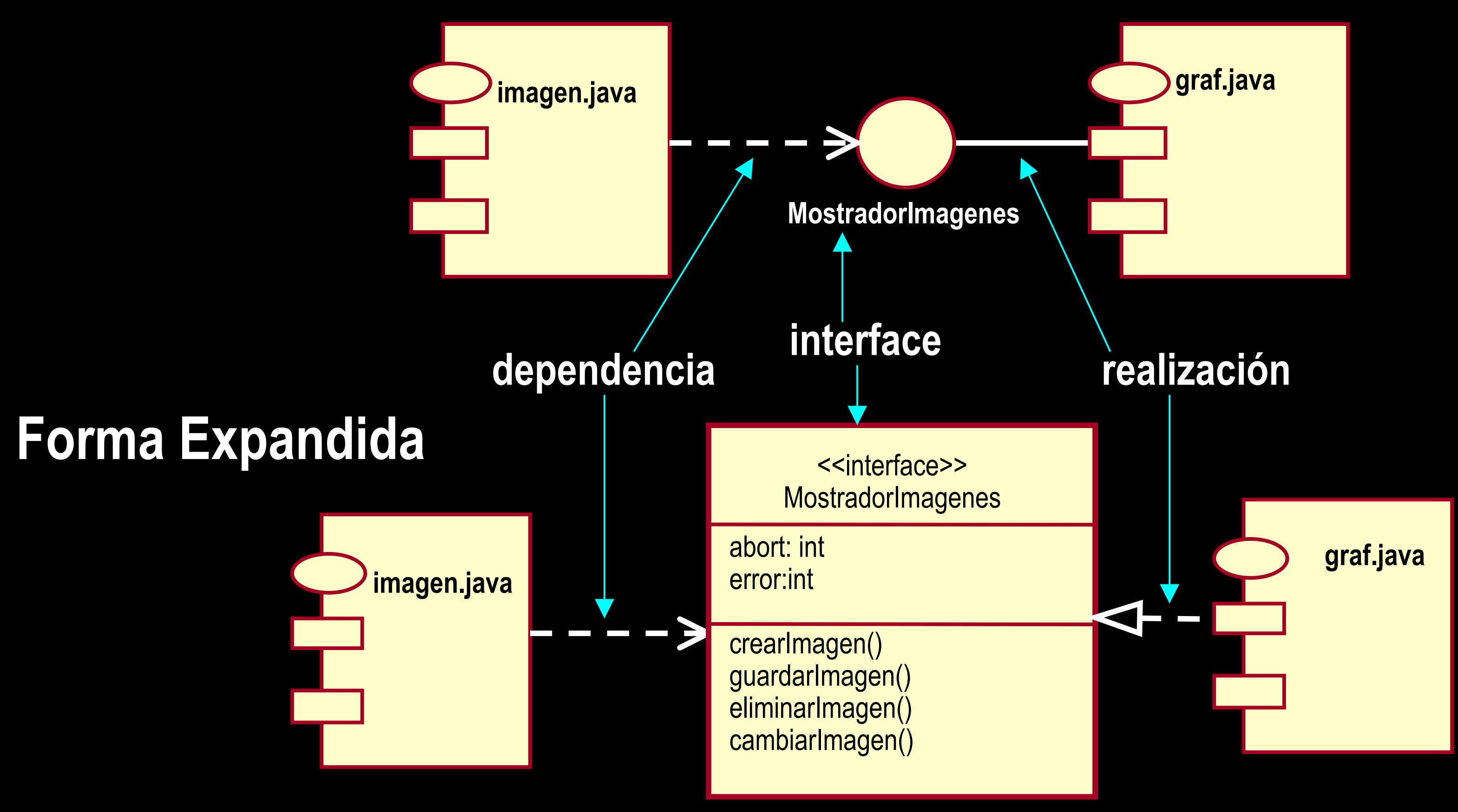


Componente de Negocios (Paquete)



Representación de Interfaces de Componentes con UML

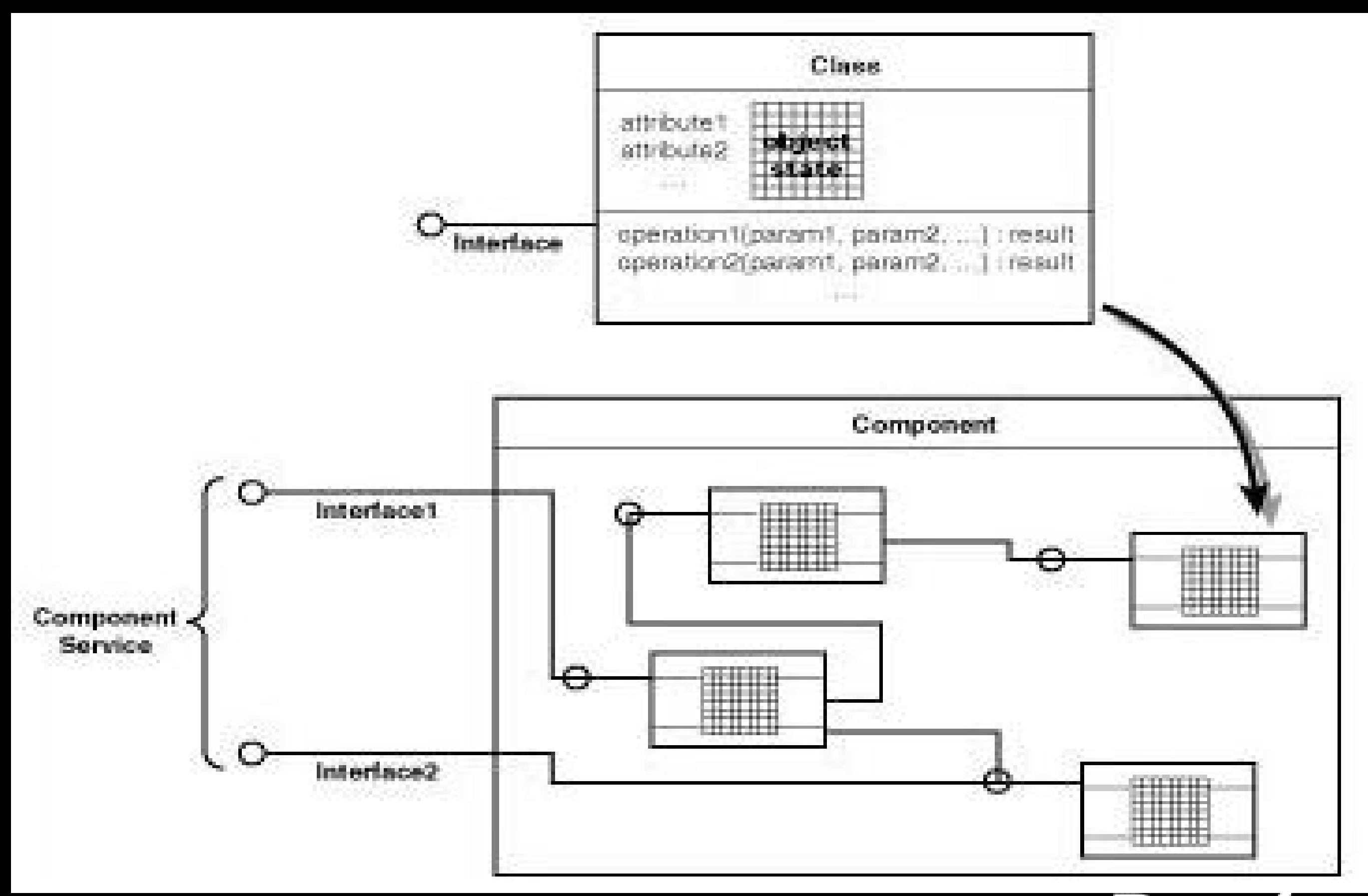
Forma Icónica



Forma Expandida

¿Qué Incluye un Componente?

- La granularidad mínima de un componente es equivalente a una clase, pero normalmente esta compuesto de mas de una.



Diagramas de Componentes

- **Al mostrar las relaciones de dependencia entre componentes, lo que se busca es lo siguiente:**
  - **Mostrar la dependencia entre componentes que representan lo mismo, pero que tienen enfoques distintos (dependencia por asignación)**
    - clases y paquetes a componentes runtime
    - código fuente a componentes runtime
    - clases y paquetes a componentes runtime a componentes de negocios
  - **Mostrar la dependencia entre componentes con el mismo enfoque**
    - dependencias de compilación (entre componentes de runtime)
    - dependencias de operación (entre componentes de runtime y de negocios)

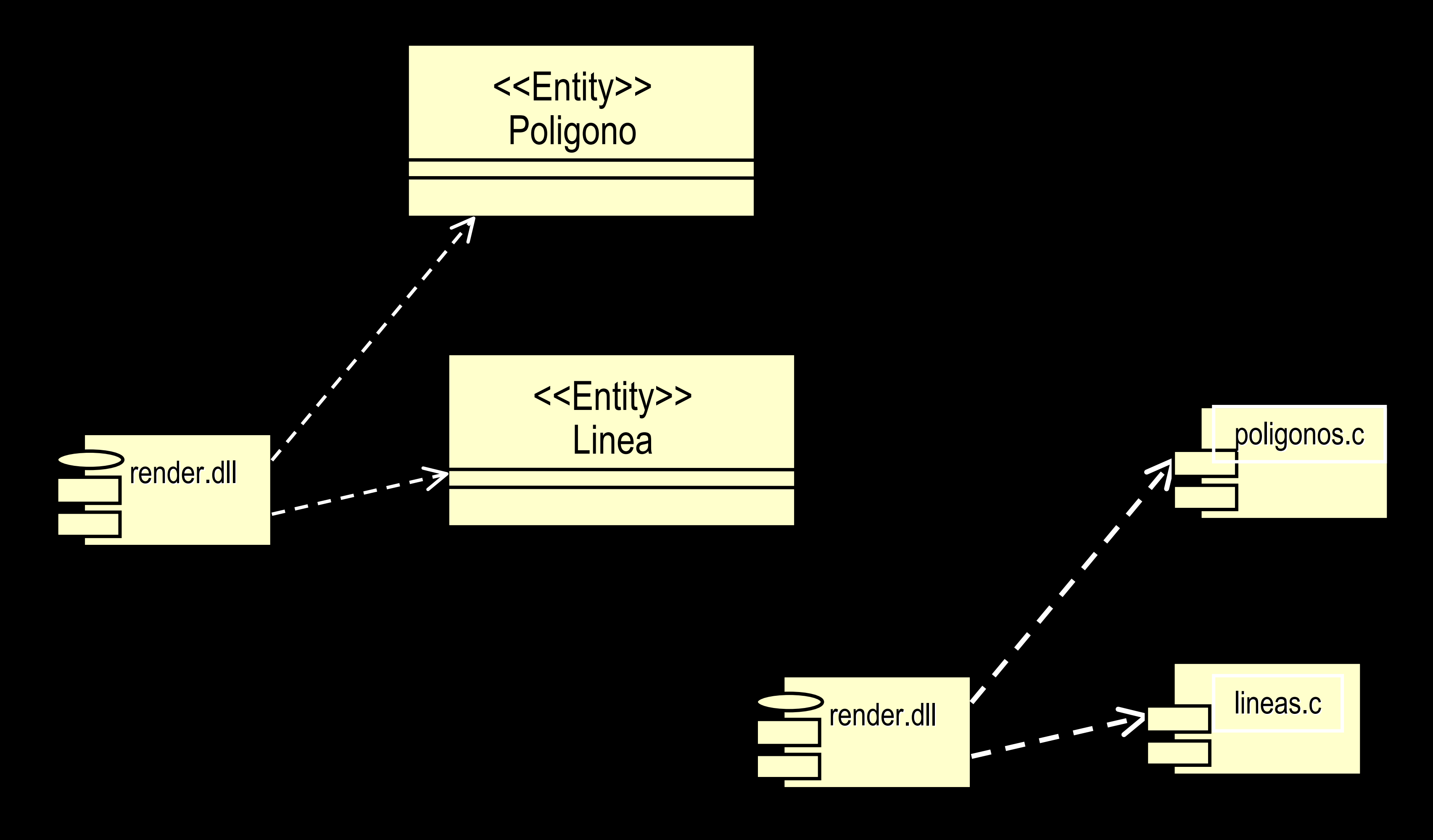


Convenciones para Diagramas de Componentes

- **Se requiere un nombre para cada componente; este nombre típicamente denota el nombre sencillo del archivo físico correspondiente en el ambiente de desarrollo**
- **Tanto la asignación como la dependencia se representan con una única relación. Una dependencia, que se representa con una flecha de línea intermitente apuntando hacia el cual componente o clase sobre el cual existe la dependencia**
- **En C++, las dependencias de compilación se indican con la directiva #include, en Java con la directiva import**

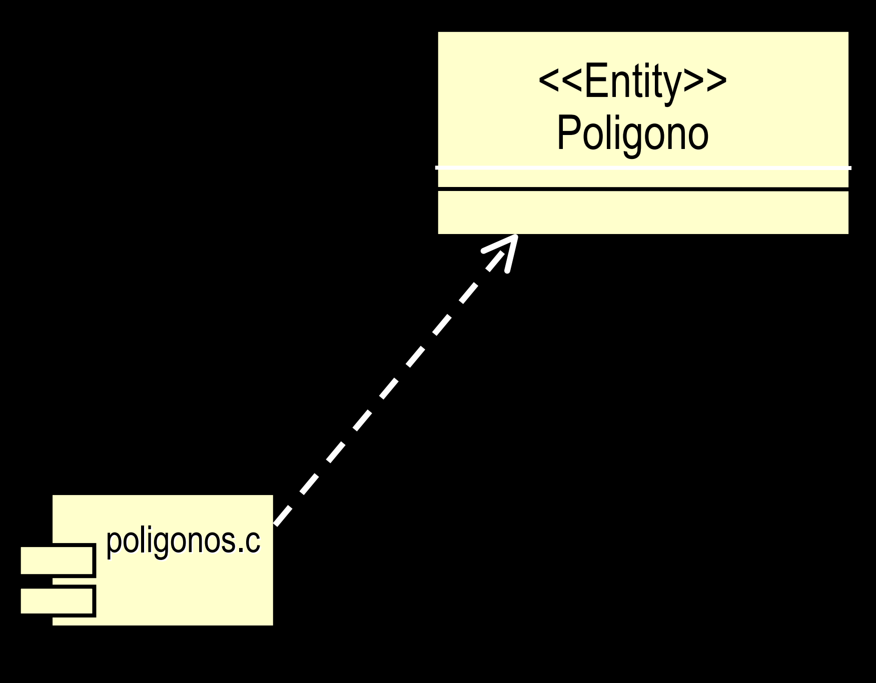
Diagramas de Componentes

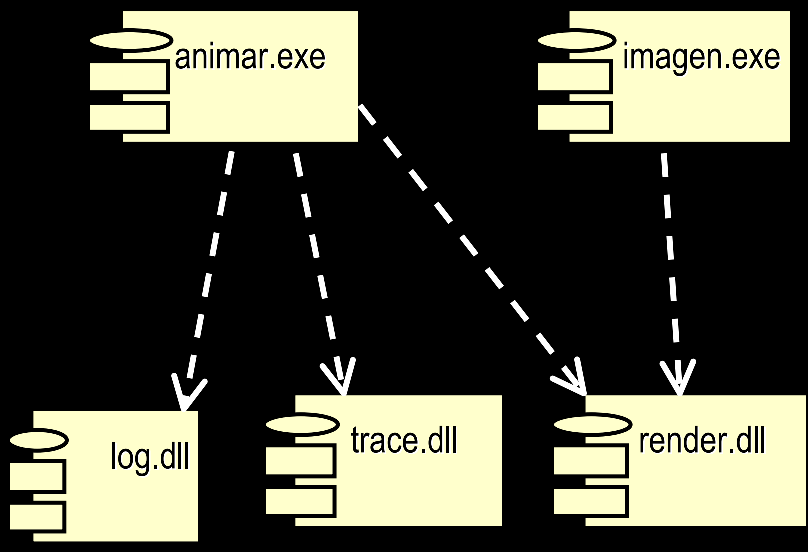
- **Mostrando Dependencia por Asignación**



Diagramas de Componentes

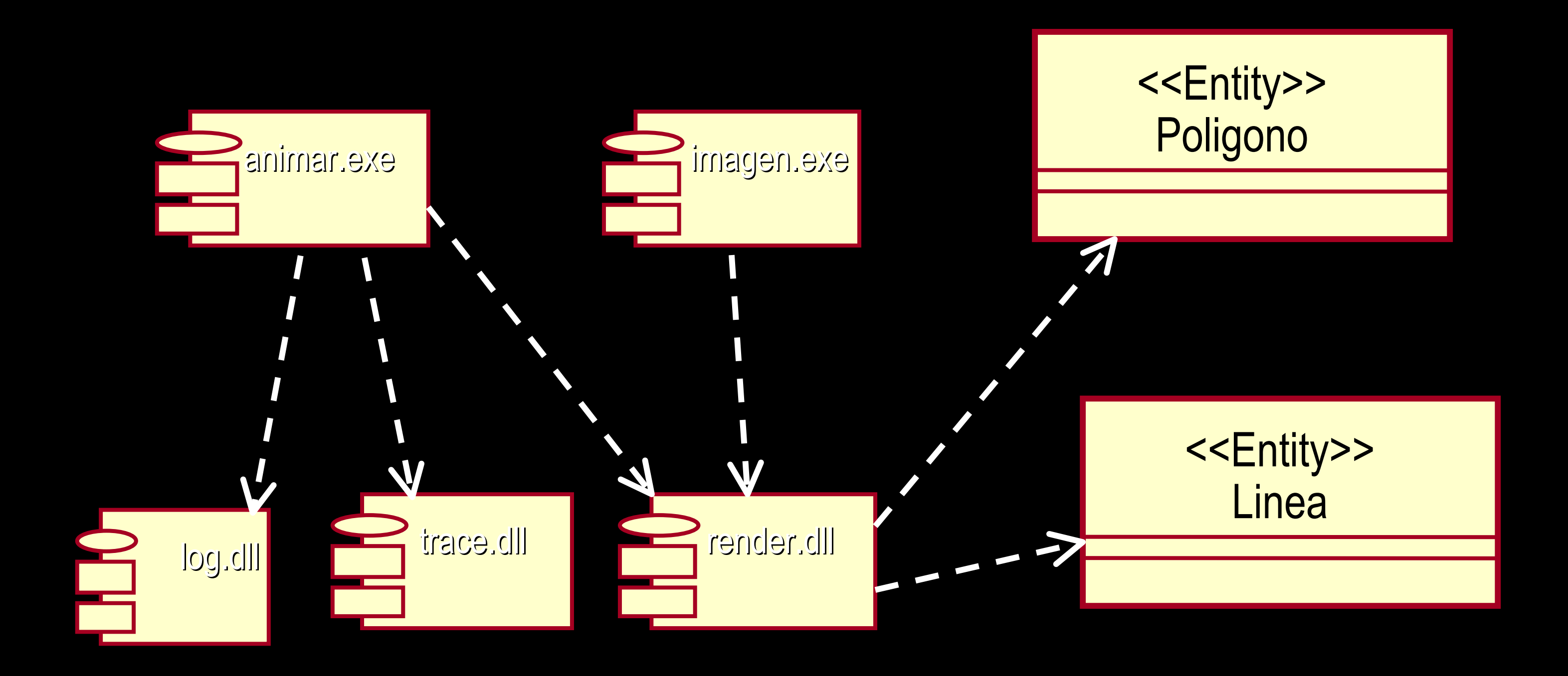
- **Mostrando Dependencia por Compilación**



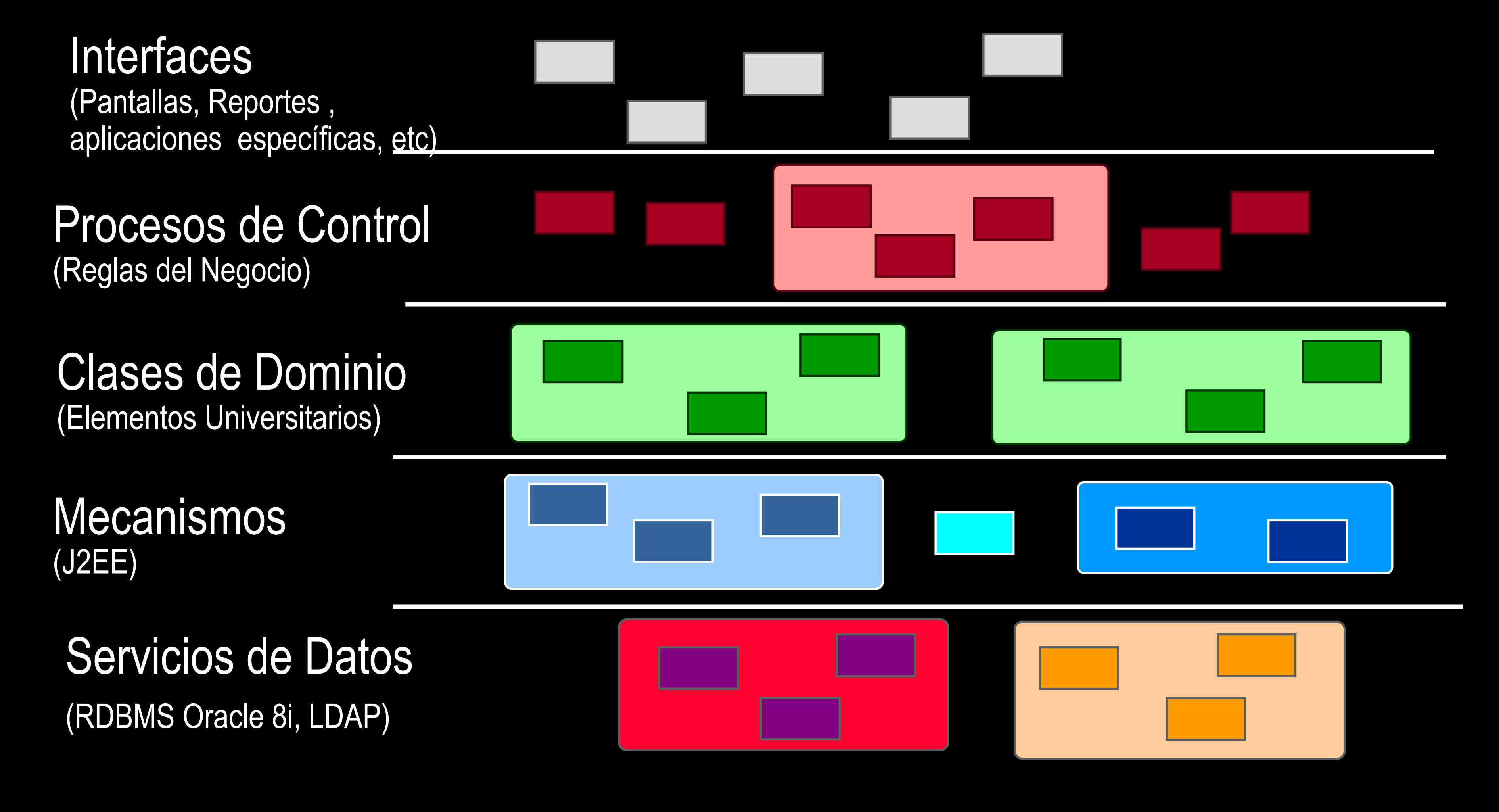


Diagramas de Componentes

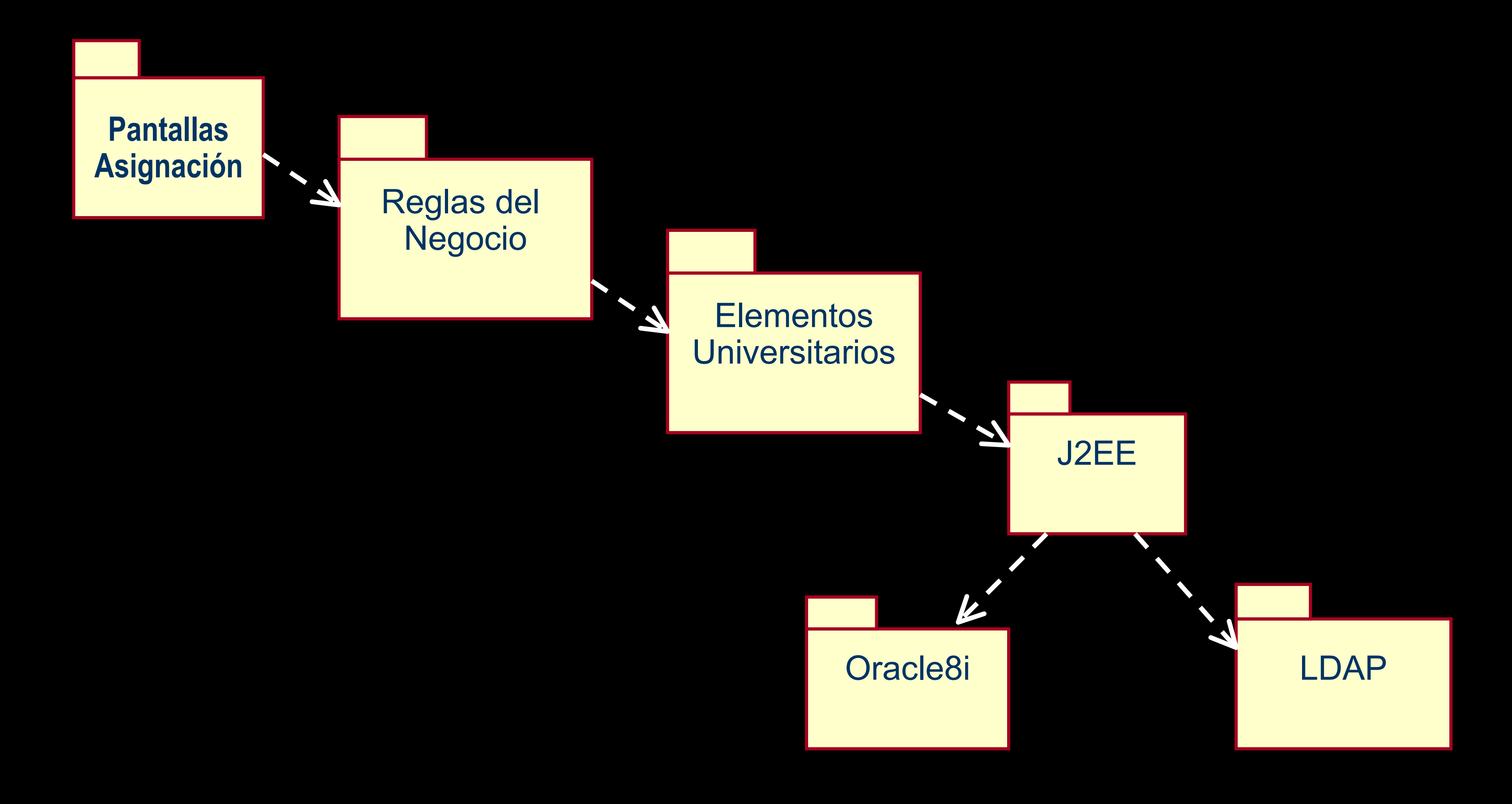
- **Dependencia por Asignación y Compilación en el mismo diagrama**



Organización de Componentes en la Arquitectura de Software del Sistema de Registro Estudiantil



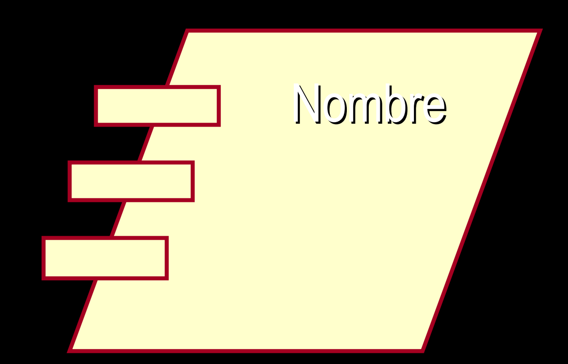
Representación de la Organización de la Arquitectura con Paquetes de UML



Vista de Procesos

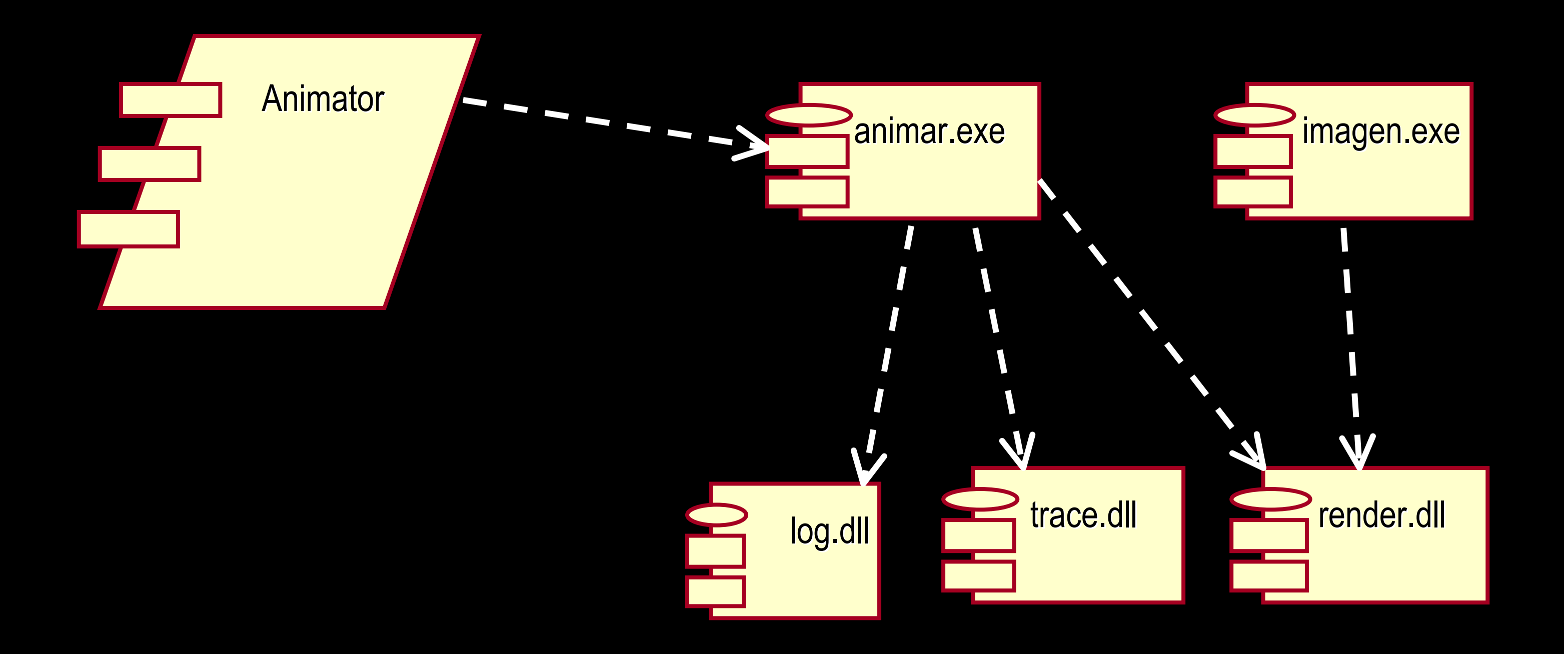
- **La vista de procesos se ocupa de la disponibilidad del sistema, su confiabilidad, rendimiento y sincronización**
- **La vista de procesos se enfoca en la descomposición de los procesos y muestra la asignación de componentes a procesos en diagramas de componentes**

Especificación de Proceso en UML



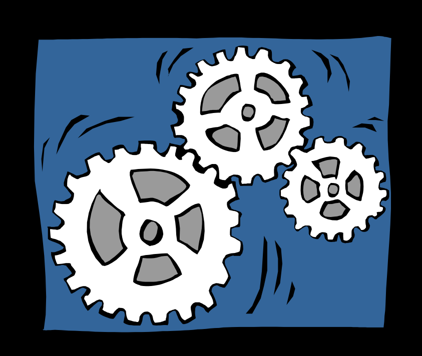


Diagramas de Componentes para la Vista de Procesos



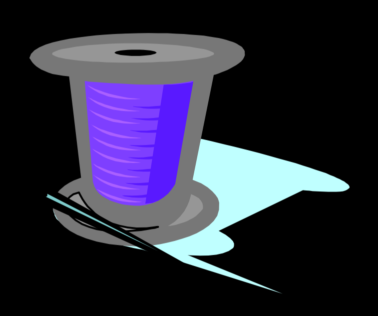
Conceptos: Proceso e Hilo (Thread)

- Proceso
  - Flujo de Control pesado – “*Heavyweight*”
  - Procesos son independientes
  - Pueden estar divididos en hilos individuales





- Thread
  - Flujo de control liviano - “*Lightweight*”
  - Los hilos corren en un contexto de un proceso





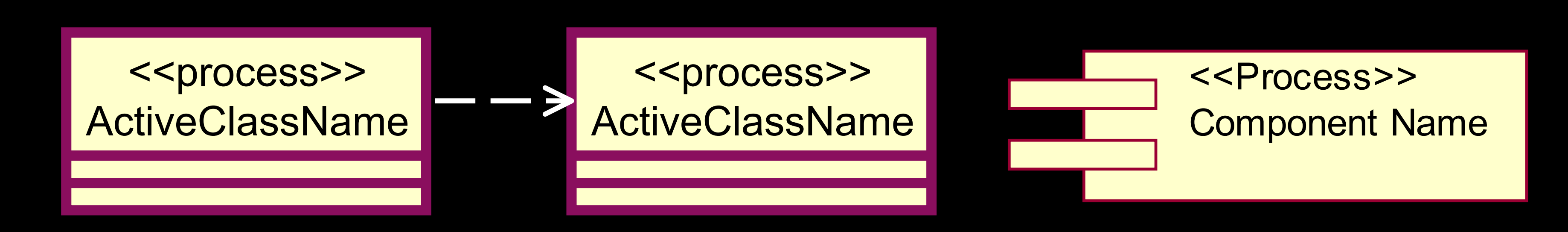
Identificando Procesos y Threads

- Para cada flujo de control separado necesitado por el sistema cree un proceso o hilo
  - Hilos de control separados pueden ser necesarios para:
    - Uso de múltiples CPUs y/o nodos
    - Incrementar la utilización del CPU
    - Proveer una reacción rápida a estimulos externos
    - Servir a eventos relacionados con el tiempo
    - Priorizar actividades
    - Escalabilidad (compartir la carga)
    - Mejorar la disponibilidad del sistema
    - Soportar los principales subsistemas

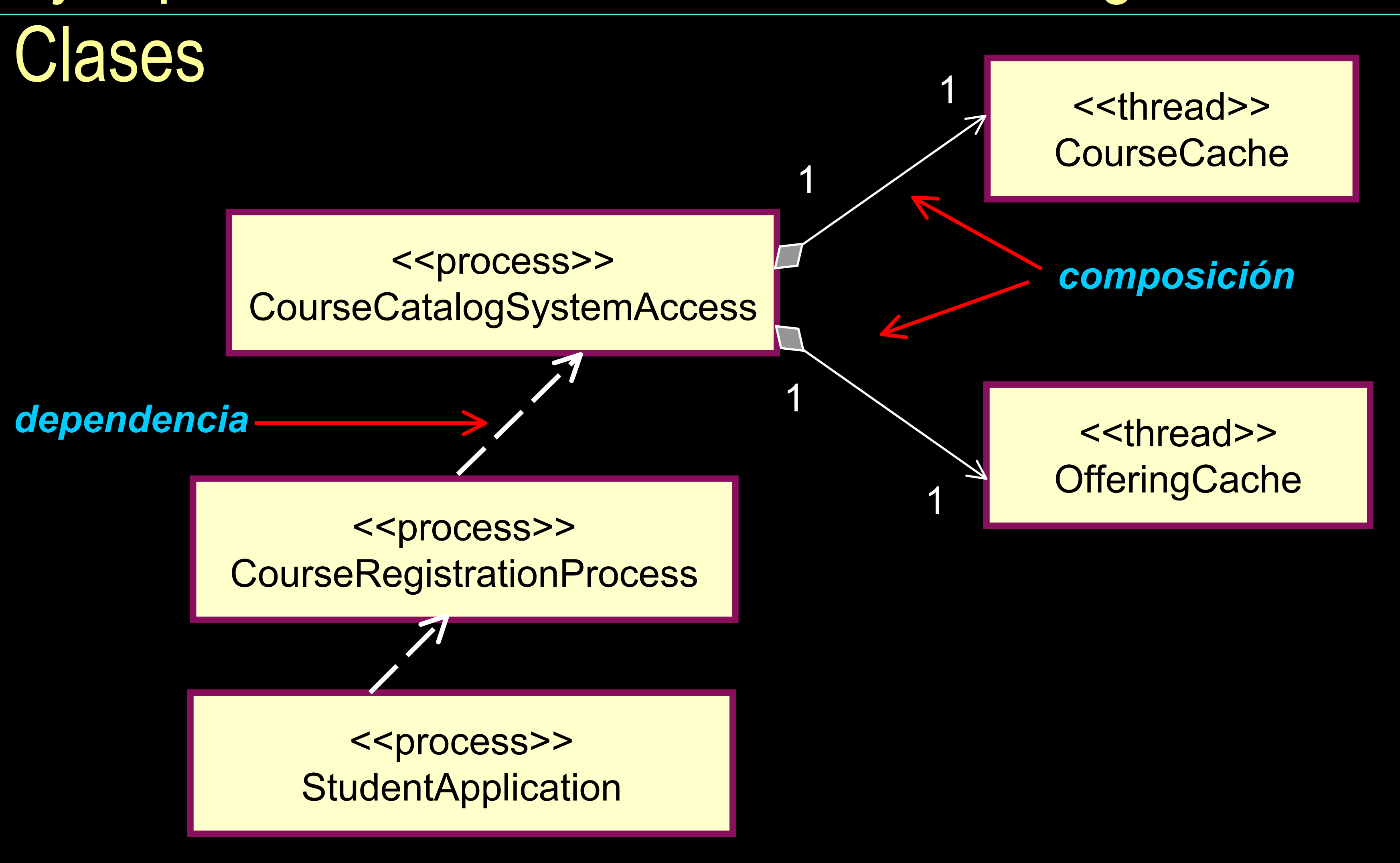


Modelando Procesos

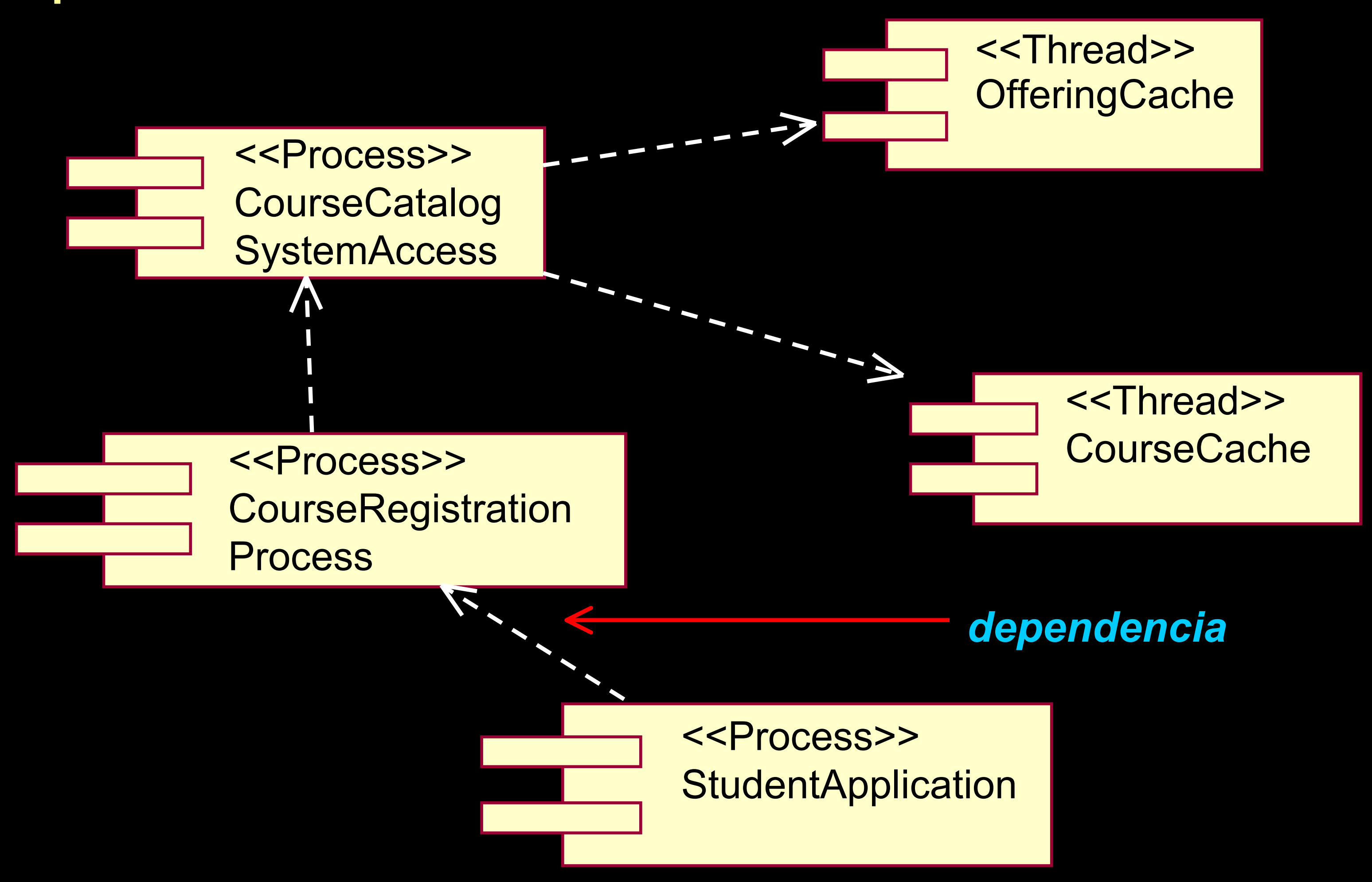
- Los procesos pueden ser modelados utilizando
  - Clases (Diagramas de clases) y Objetos (Diagramas de Interacción)
  - Componentes (Diagramas de Componentes)
- Estereotipos: <<process>> o <<thread>>
- La relación entre procesos puede ser modelada como una dependencia



Ejemplo: Modelando Procesos con Diagramas de Clases



Ejemplo: Modelando Procesos con Diagramas de Componentes



Vista de Producción

- La vista de producción debe mostrar la distribución física de la carga de procesamiento, asociando los procesos de componentes de ejecución a nodos específicos
- Para hacer esta distribución se toman en consideración requerimientos tales como rendimiento, funcionamiento y tolerancia a fallos
- La vista de producción incluye uno o mas diagramas de "Deployment"

Plataformas Tecnológicas y de Distribución

- Esta parte de la arquitectura esta orientada a definir las estructura física del sistema, en términos de cómo deben distribuirse y configurarse los componentes en nodos de procesamiento, y como pueden comunicarse estos nodos.
- Muchas veces se utilizan diagramas de bloque de hardware que utilizan esta perspectiva para representar la arquitectura de un sistema, sin embargo, no debe confundirse esta presentación con la de la Arquitectura de Software.
- El problema radica en que las “capas” que se presentan aquí están dentro del contexto de la Arquitectura Computacional de Redes (NCA) y no corresponden necesariamente a las capas de la Arquitectura de Software que se han utilizado anteriormente.

Plataformas Tecnológicas y de Distribución

- El impacto de estos elementos en la arquitectura del sistema es muy grande, y en general es el criterio mas común para definir el modelo de arquitectura. Esto obedece a que en la mayoría de los casos la arquitectura del sistema debe adaptarse a la infraestructura de hardware existente, o que se busca adoptar un modelo computacional por razones estratégicas.

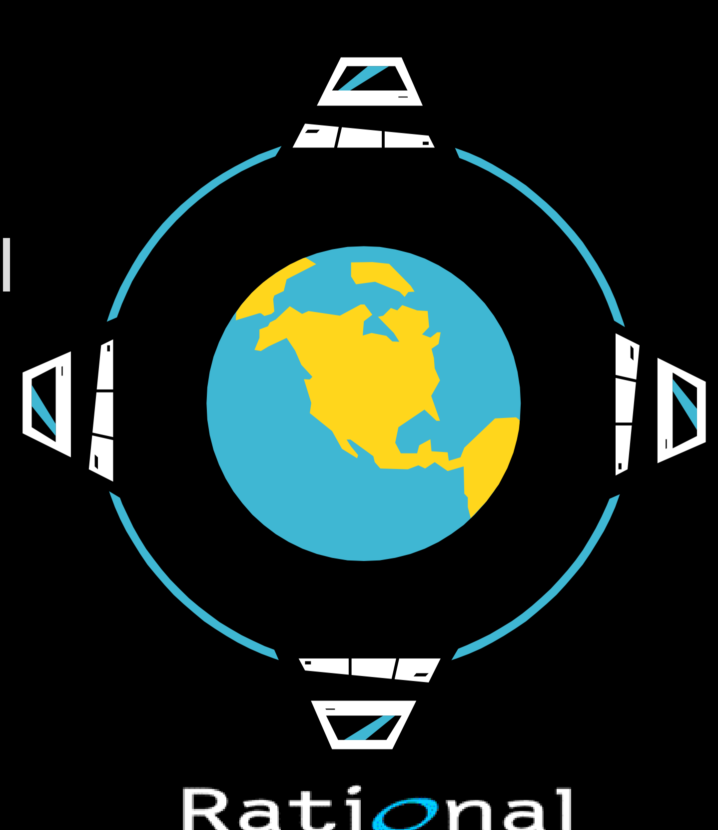
- Los ejemplos mas comunes de esto son: Procesamiento Centralizado, Peer to Peer, Cliente/Servidor y Ambientes WEB.

Plataformas Tecnológicas y de Distribución: el Network Computing Architecture

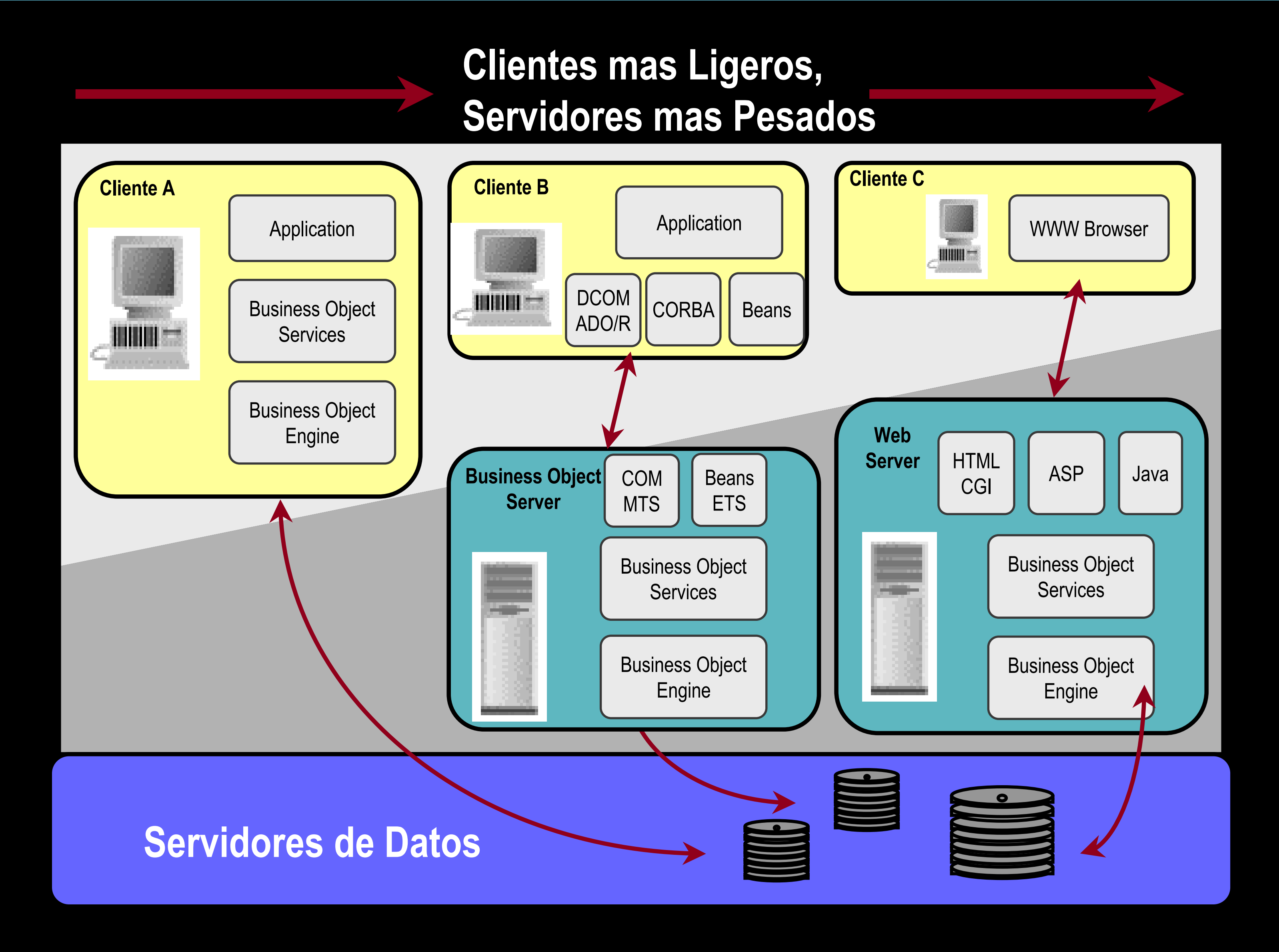
- La "Arquitectura Computacional de Redes" o NCA (Network Computing Architecture), es el modelo base del cual parte cualquier sistema que incluye procesamiento distribuido entre 2 o más nodos que se comunican entre si por una red de datos.
- **Los dos modelos básicos que describe el NCA son:**
  - "Peer-to-Peer": comunicación entre dos procesos similares para balancear o distribuir la carga de procesamiento.
  - "Cliente/Servidor": comunicación entre un proceso cliente que solicita servicios de un proceso servidor.

Patrones de Distribución

- Cliente/Servidor
  - Fat Client (Cliente pesado – Procesamiento del lado del Cliente)
  - 3-tier
  - Fat Server (Servidor pesado – Procesamiento del lado del Servidor)
  - Distributed Client/Server
- Peer-to-peer



Arquitecturas Cliente/Servidor

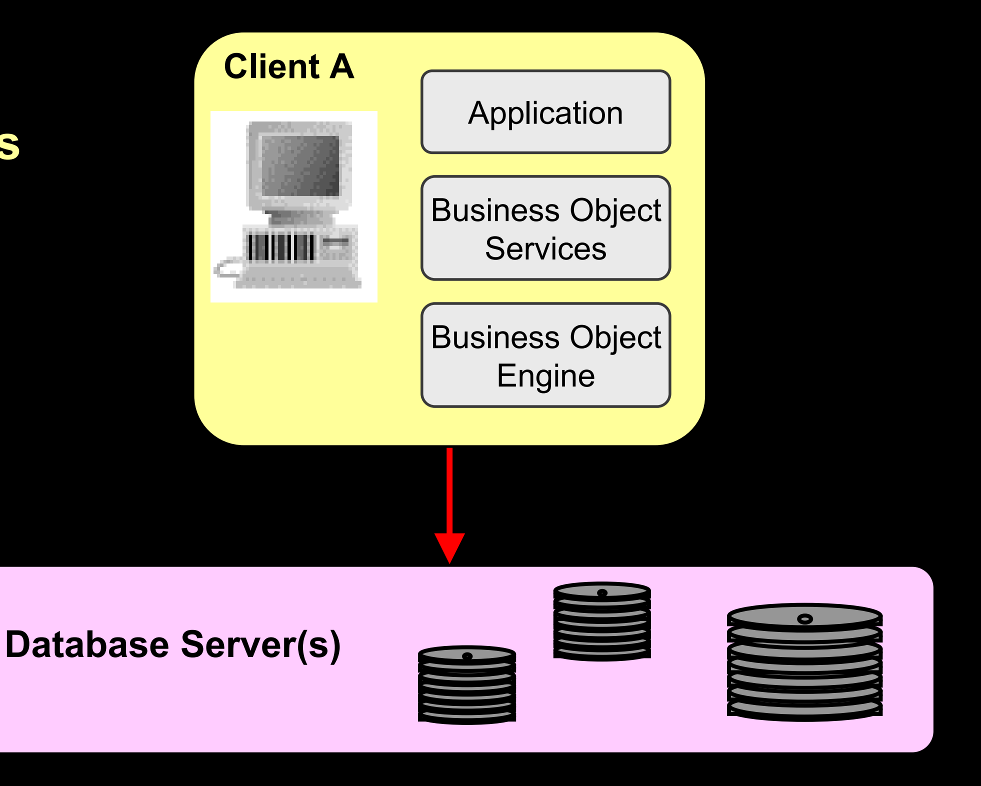


Cliente/Servidor: Arquitectura “Fat Client”

Application Services

Business Services

Data Services





Cliente/Servidor: Arquitectura Three-Tier

Application Services

Client B

Application

DCOM ADO/R

CORBA

Beans

Business Services

Business Object Server

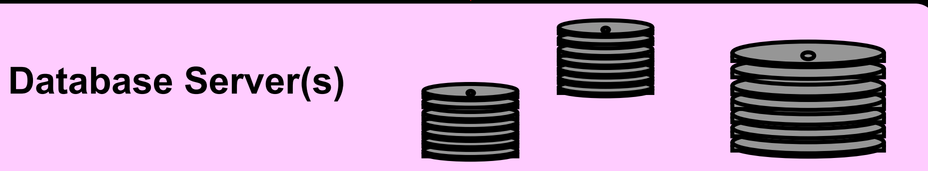
COM MTS

Beans ETS

Business Object Services

Business Object Engine

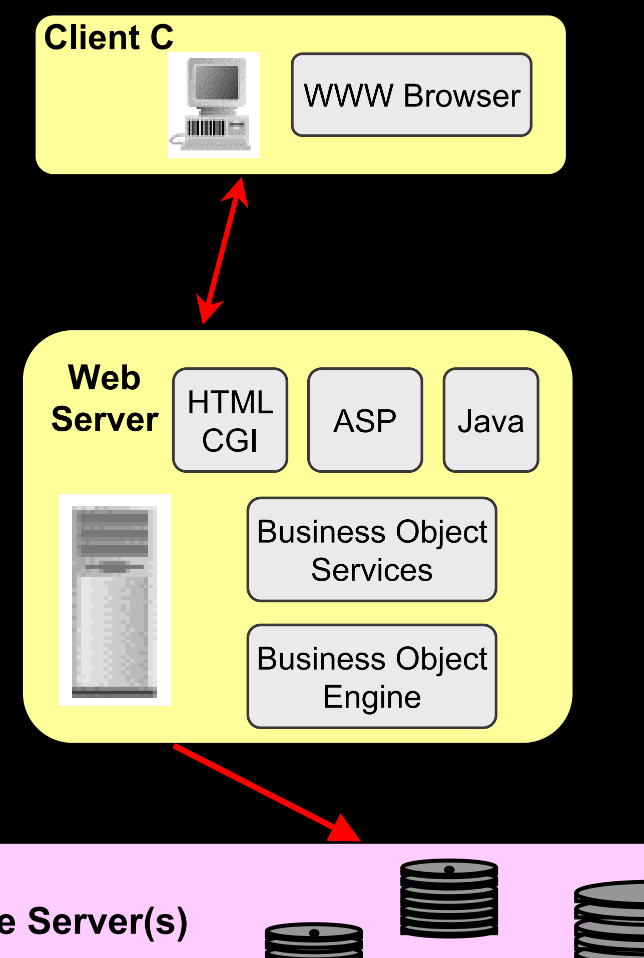
Data Services



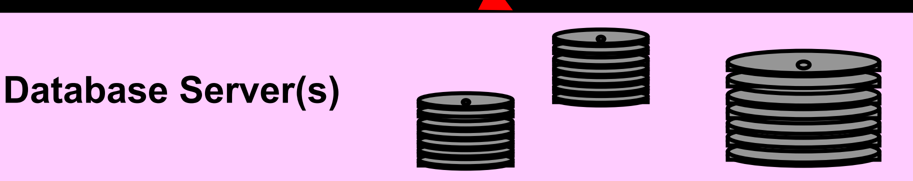
Cliente/Servidor: Arquitectura Aplicación Web

Application Services

Business Services



Data Services





Arquitectura Peer-to-Peer

Application Services

Business Services

Data Services

Application

DCOM ADO/R | CORBA | Beans

COM MTS | Beans ETS

Business Object Services

Business Object Engine

Application

DCOM ADO/R | CORBA | Beans

COM MTS | Beans ETS

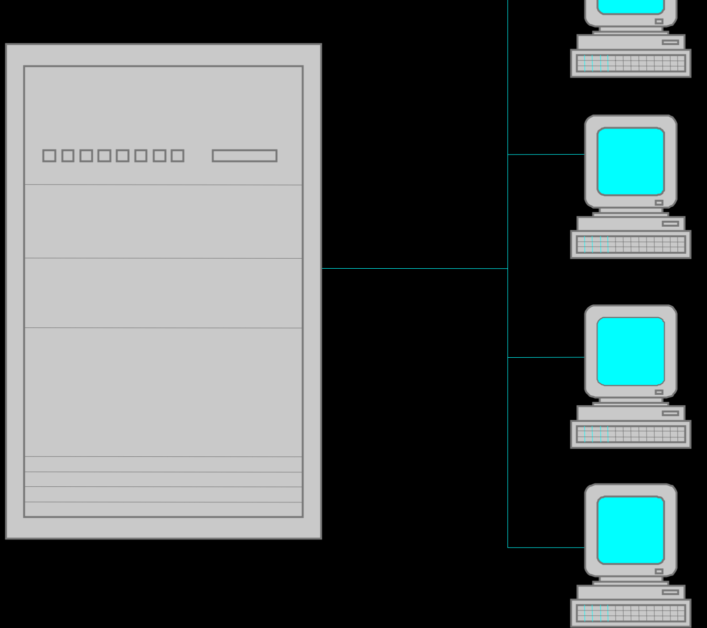
Business Object Services

Business Object Engine



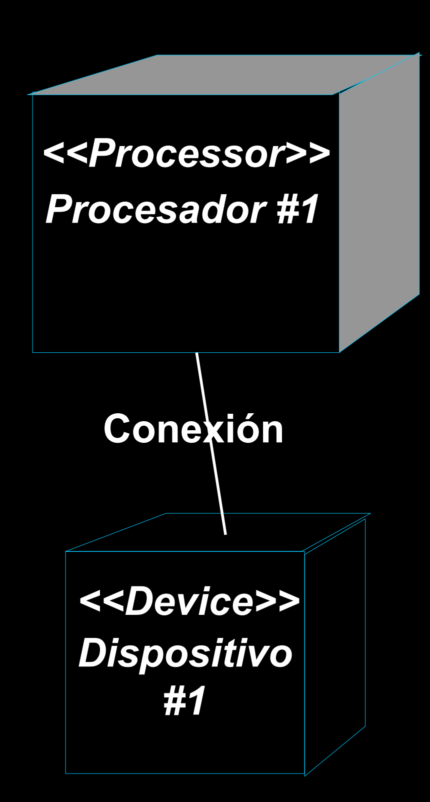
Configuración de Red

- Estaciones de Trabajo de Usuario Final
- Servidores de procesamiento
- Configuraciones
  - Desarrollo
  - Pruebas
- Procesadores Especializados

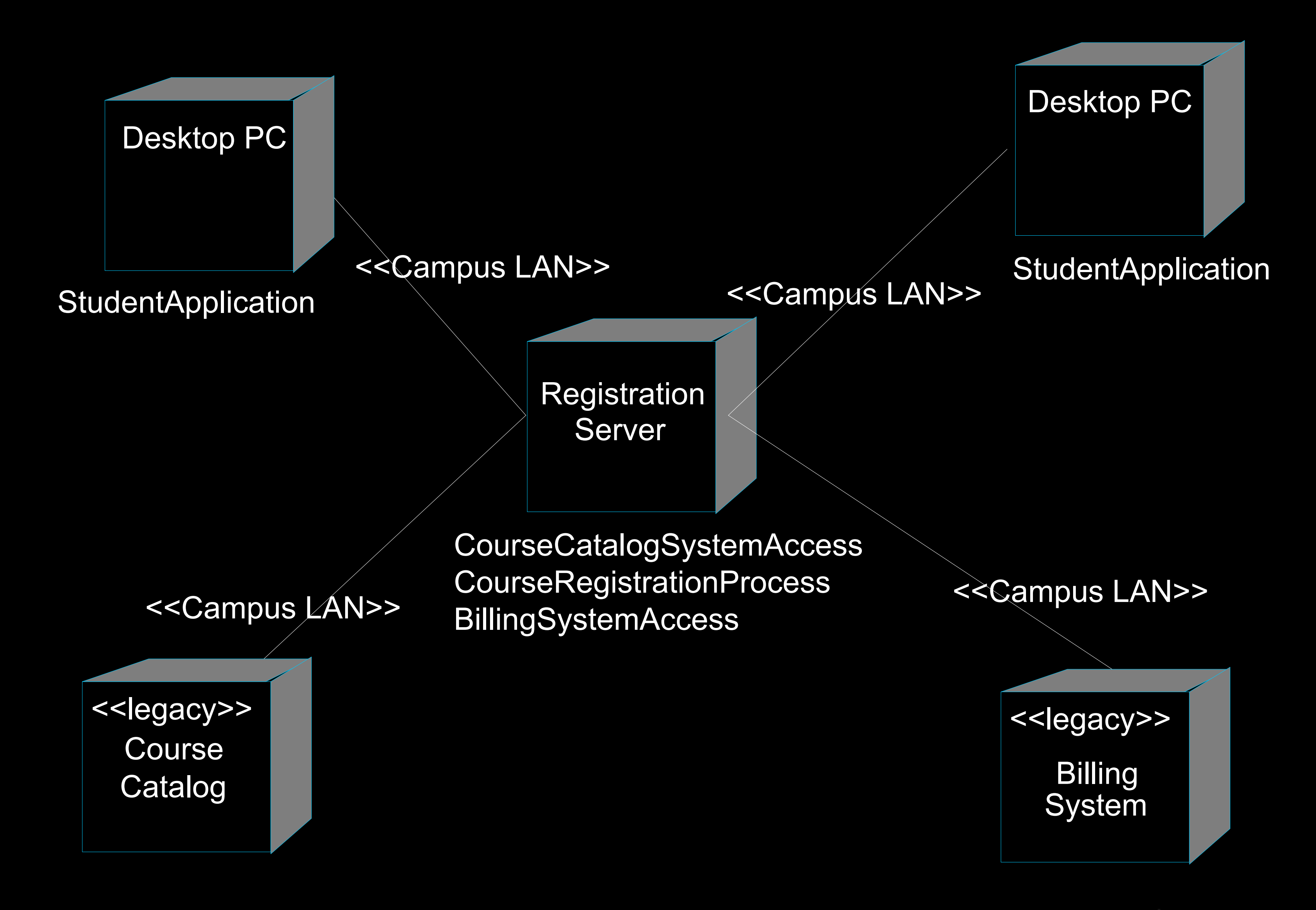


Elemento de Modelación para Producción

- Nodo
  - Recurso físico conputacional de tiempo de ejecución
  - Nodo Procesador (Processor Node)
    - Ejecuta Software del Sistema
  - Nodo Dispositivo (Device Node)
    - Dispositivo de Soporte
    - Controlado típicamente por un procesador
- Conexión
  - Mecanismo de Comunicación
  - Medio Físico
  - Protocolo de Software



Ejemplo: Asignación de Procesos a Nodos



Diseño de la Arquitectura de Software

Diseño de la Arquitectura de Software

- **El diseño de la arquitectura de software de un sistema tiene que enfocarse hacia el manejo de riesgos.**
  - Debe orientarse principalmente a resolver los Requerimientos de Arquitectura
  - Se realiza mejor a través de un proceso de desarrollo iterativo e incremental que permita validar las decisiones de arquitectura
- **En el diseño de la arquitectura se deben resolver los Requerimientos de Arquitectura con la implementación de uno o pocos patrones de comportamiento común, que puedan ser alcanzados a través de abstracciones y mecanismos que puedan usarse en común y/o que sean parte de una infraestructura estándar (Mecanismos de Arquitectura).**
- **Debe ser realizado por un Arquitecto, o un equipo de arquitectura dirigido por un Arquitecto Principal.**

Requerimientos de Arquitectura

- **Son todos los requerimientos de alto riesgo, alta prioridad y/o baja estabilidad; es decir, los que pueden poner en peligro el proyecto**
- **Pueden ser tanto funcionales como no funcionales, pero además pueden considerarse como:**
  - Explícitos: son los que están establecidos o descritos explícitamente en los requerimientos; por ejemplo, casi todos los requerimientos no funcionales contenidos en las Especificaciones Suplementarias son explícitos
  - Implícitos: son los que pueden derivarse o resultar de los requerimientos, ya que implícitamente están contenidos en ellos; por ejemplo, para notificar con un correo electrónico se necesita contar con la infraestructura apropiada

Clasificación de los Requerimientos de Arquitectura

- **Los Requerimientos de Arquitectura, sean explícitos o implícitos, deben clasificarse adecuadamente para poder resolverlos; una técnica muy efectiva es el uso del Sistema FURPS+ desarrollado por Robert Grady en Hewlett Packard.**
- **FURPS+ es un acrónimo en Ingles que significa:**
  - Functionality (Funcionalidad)
  - Usability (Usabilidad)
  - Reliability (Confiabilidad)
  - Performance (Rendimiento)
  - Supportability (Soportabilidad)



El Sistema FURPS+

- Funcionalidad: estos requerimientos generalmente representan las características principales o servicios que el sistema puede brindar e incluye los requerimientos funcionales implícitos y explícitos.
- Usabilidad: son los requerimientos relacionados con las características del interface de usuario, como aspectos estéticos, facilidad de uso y consistencia.
- Confiabilidad (Reliability): se refiere a características como disponibilidad (tiempo que el sistema debe estar disponible), exactitud (de resultados en cálculos, etc.) y robustez (capacidad del sistema de recuperarse de una falla).
- Rendimiento (Performance): se refiere a características como capacidad de proceso (en transacciones por segundo, etc.), tiempo de respuesta, tiempo de recuperación y tiempo de encendido y apagado.

El Sistema FURPS+

- **Soportabilidad: son los requerimientos relacionados con características como capacidad de prueba (testability), adaptabilidad, mantenibilidad, compatibilidad, configurabilidad, instalabilidad, escalabilidad y localidad (capacidad de utilizarse en diferentes locaciones, idiomas, etc).**
- **El "+" del acrónimo FURPS+ se refiere a categorías que normalmente implican restricciones de:**
  - Diseño: especifican opciones de diseño; por ejemplo, debe usarse un RDBMS.
  - Implementación: establece restricciones de cómo programas o construir el sistema; por ejemplo, lenguajes de programación y limites del sistema.
  - Interface: especifican elementos externos con los que el sistema debe interactuar, incluyendo estándares, formatos y demás factores relacionados.
  - Físicas: se refieren a restricciones físicas que debe cumplir el hardware en donde funcionara el sistema; por ejemplo, tamaño, peso, forma, etc.

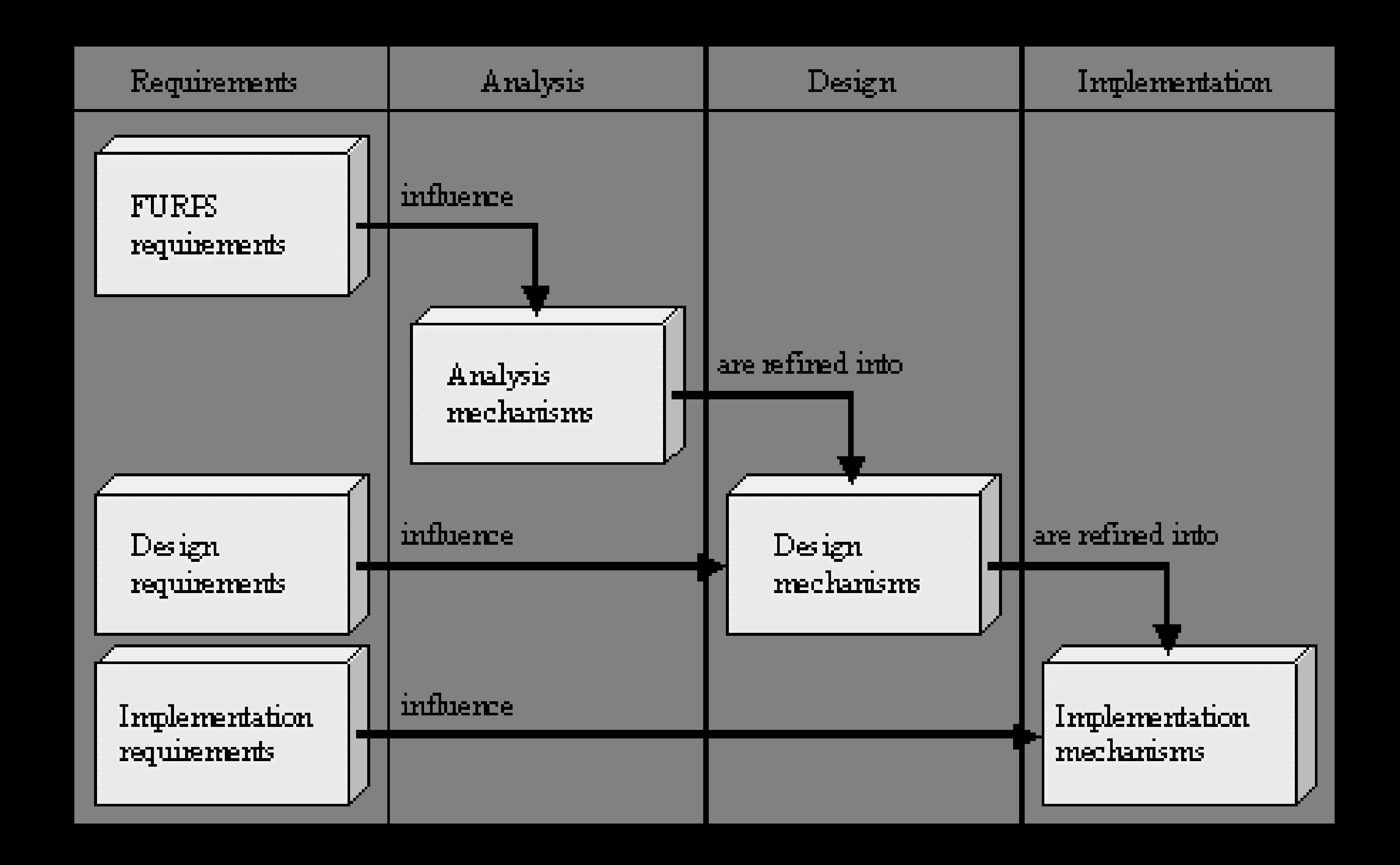
Beneficios de usar el Sistema FURPS+

- Establece un contexto o guía para la definición, especificación y clasificación sistemática de los requerimientos de Arquitectura
- Facilita establecer relaciones entre requerimientos que aparentemente no están relacionados.
- Facilita determinar si todos los requerimientos de arquitectura están considerados y/o se han completado.
- Mejora la comunicación con los afectados (stakeholders) y usuarios, ya que facilita definir las preguntas apropiadas para detallar los requerimientos.

Mecanismos de Arquitectura

- Son las abstracciones y mecanismos (de preferencia estandarizados) que se utilizan dentro de un patrón de diseño para resolver los Requerimientos de Arquitectura.
- Los mecanismos de arquitectura se pueden clasificar en tres categorías:
  - Mecanismos de Análisis (conceptual)
  - Mecanismos de Diseño (concreto)
  - Mecanismos de Implementación (especifico)

FURPS+ y los Mecanismos de Arquitectura



Mecanismos de Análisis

- Capturan los principales aspectos de una solución de manera que es independiente de la implementación.
- Proveen comportamientos específicos de una clase relacionada a un dominio.
- Ejemplos de mecanismos son: el manejo de persistencia, comunicación entre procesos, seguridad, distribución.
- Por ejemplo en análisis se determinan los mecanismos que aplican a ciertas clases, de esta manera:
  - **Curso necesita manejo de Persistencia**
  - **SistemaFacturacion necesita manejar Comunicación entre Procesos**
  - **Estudiante necesita manejar Persistencia y Seguridad**

Mecanismos de Diseño

- Definen opciones concretas para los mecanismos de análisis.

- Asumen algunos detalles del ambiente de implementación, pero no se enlazan a una implementación específica (tal y como lo hacen los mecanismos de implementación).

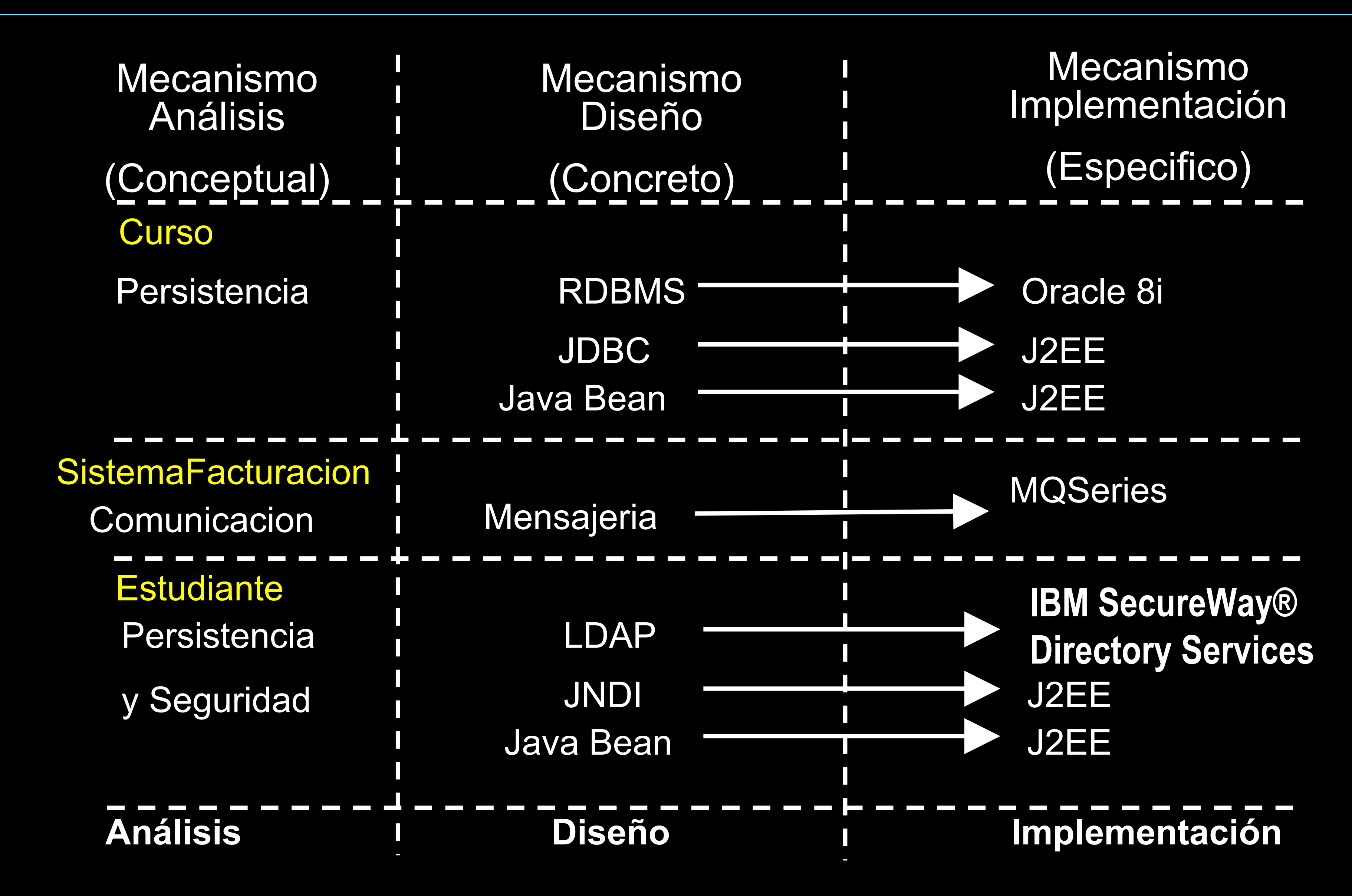
Mecanismos de Diseño

- Por ejemplo en el diseño se establece como los mecanismo de análisis se van a resolver con mecanismos de diseño de esta manera:
  - **Curso debe hacerse Persistente y para ello se utilizara un Java Bean, JDBC y un RDBMS.**
  - **SistemaFacturacion necesita manejar Comunicación entre Procesos y para ello se utilizara una cola Mensajería con archivos de texto.**
  - **Estudiante debe hacerse Persistente y para ello se utilizara un Java Bean y en un repositorio LDAP, ademas debe permitir manejar Seguridad con un servicio de autenticación que utilice el repositorio LDAP y JNDI.**

Mecanismos de Implementación

- Especifican de manera exacta la implementación de los mecanismos de diseño.
- Están ligados a cierta tecnología, lenguaje de programación, proveedor, etc.
- Por ejemplo en el diseño se establece como los mecanismos de diseño se van a resolver con mecanismos de implementación de esta manera:
  - **Curso debe hacerse Persistente y para ello se utilizara un Java Bean y JDBC de J2EE, y un RDBMS que será Oracle 8i.**
  - **SistemaFacturacion necesita manejar Comunicación entre Procesos y para ello se utilizara una cola de Mensajería con archivos de texto que se implementara con MQSeries usando el protocolo SOAP sobre archivos XLM.**
  - **Estudiante se hará persistente y para ello se utilizara un Java Bean (de J2EE) y en un repositorio LDAP que será IBM SecureWay® Directory Services, además debe permitir manejar Seguridad con un servicio de autenticación que utilice el repositorio LDAP y JNDI (de J2EE).**

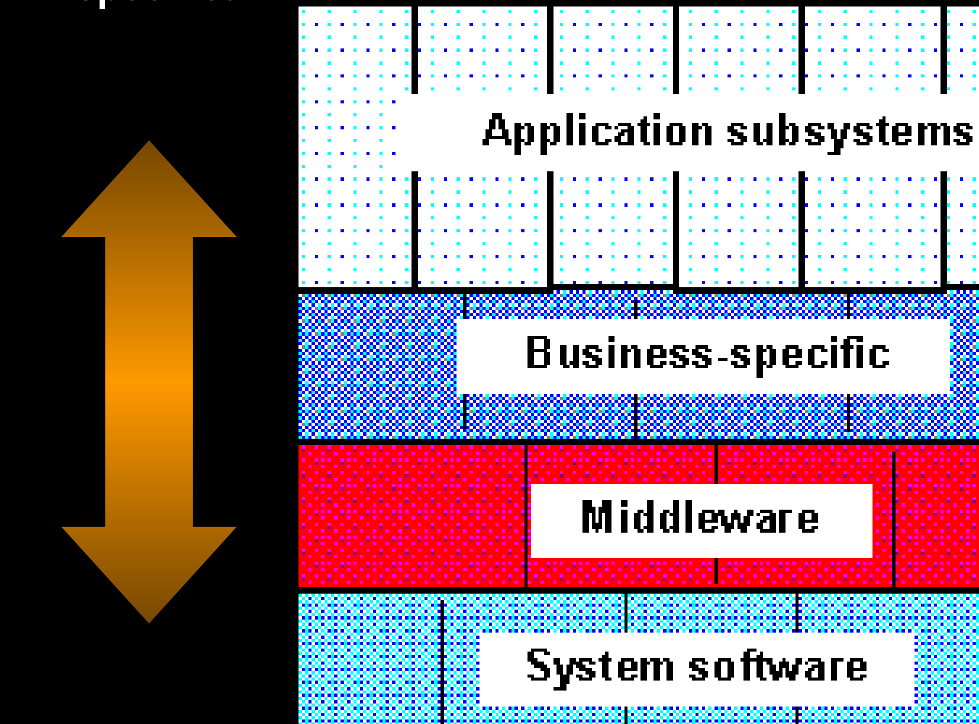
Mecanismos de Diseño e Implementación



Capas de Arquitectura

- Establecen un modelo para organizar y estructurar los Mecanismos de Arquitectura según los patrones de diseño que se hallan utilizado para resolver los Requerimientos de Arquitectura
- En términos generales, todo modelo de capas es una especialización del "Modelo Genérico de Capas de Arquitectura"
- Idealmente deben definirse según los criterios ya mencionados para utilizar componentes y lograr los beneficios de las Arquitecturas basadas en Componentes

Modelo Genérico de Capas de Arquitectura



Distinct application subsystem that make up an application - contains the value adding software developed by the organization.

Business specific - contains a number of reusable sybsystems specific to the type of business.

Middleware - offers subsystems for utility classes and platform-independent services for distributed object computing in heterogeneous environments and so on.

System software - contains the software for the actual infrastructure such as operating systems, interfaces to specific hardware, device drivers and so on.



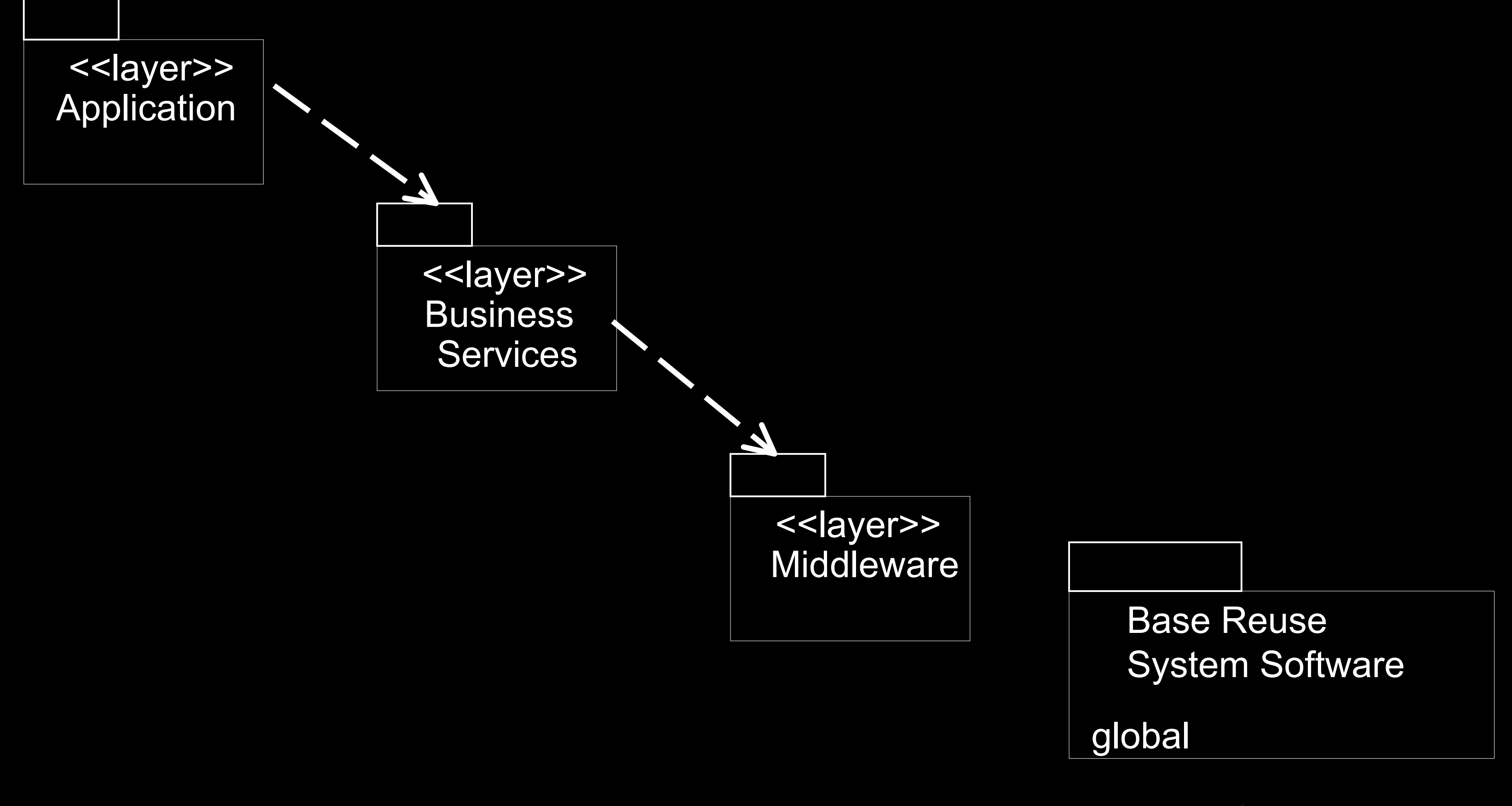
Guia para Definir Capas

- **Visibilidad**
  - Las dependencias deben darse solo en la misma capa y capas inferiores
- **Volatilidad**
  - Las capas superiores son afectadas por cambios en los requerimientos
  - Las capas inferiores son afectadas por cambios en la configuración
- **Generalidad**
  - Elementos de modelo mas abstractos en las capas más bajas
- **Número de Capas**
  - Sistemas Pequeños: 3-4 capas
  - Sistemas Complejos: 5-7 capas

El objetivo es reducir los acoplamientos para facilitar el mantenimiento



Modelo Generico de Capas de Arquitectura Representado con UML



Clasificación de las Arquitecturas de Software según su Modelo o Estilo

Modelos (Estilos) de Arquitectura de Software

Un Modelo o Estilo, es un propiedad de la Arquitectura de Software que limita las opciones disponibles para estructurarla y le impone un grado de uniformidad.

Elementos que Definen un Modelo o Estilo de Arquitectura de Software

- Un Modelo de Arquitectura de Software Puede estar definido por la selección de uno o más de los siguientes elementos:
  - **Patrones de Arquitectura (Arquitectural Patterns):** Batch, Model-View-Controller (MVC), Factory, Mediador, Proxy, etc.
  - **Mecanismos e Infraestructuras de Arquitectura (Arquitectural Frameworks and Mechanisms):** JDBC, EJB, J2EE, DCOM, .NET, etc.
  - **Plataformas Tecnológicas y de Distribución**: Procesamiento Centralizado, Peer to Peer, Cliente/Servidor, Ambientes WEB, etc.
  - **Técnicas de Descripción de la Arquitectura:** A&D Estructurado, UML
  - **Herramientas o productos utilizados para Implementar la Arquitectura:** Java, Visual Studio, RDBMS & SQL, etc.

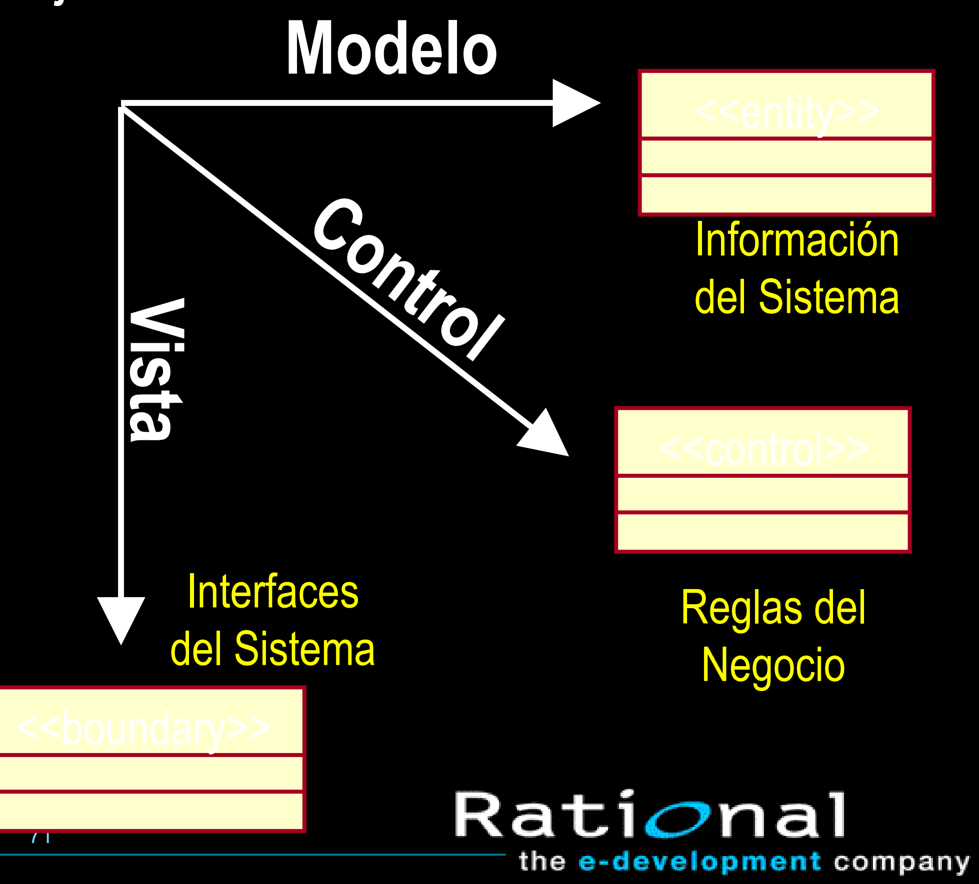
Patrones de Arquitectura

- Un patrón de arquitectura es un solución probada para un problema común que se da en una situación especifica de diseño; es otra de las analogías con la arquitectura civil, donde hay patrones de diseño para edificios, puentes, casas, etc.
- Los patrones de arquitectura existen a un nivel de abstracción mas alto que los componentes; sirven por ejemplo para organizar las capas de la arquitectura.
- El patron mas comun que se utiliza en arquitecturas basadas en componentes es el Model-View-Controller (MVC). Este patrón es la base detrás del “Modelo de 3 Capas”, pero su interpretación a nivel de Arquitectura de Software a menudo se confunde en funcion de la distribucion de nodos de procesamiento según la Arquitectura Computacional de Redes (Network Computer Architecture - NCA).

El Patrón MVC

Es ta implícito en el Análisis de Casos de Uso y usa clases estereotipadas según la metodología OOSE de Ivar Jacobson para representar modelos ideales de objetos

- Los componentes se definen enfocados a la separación de responsabilidades, para facilitar la reutilización y extensibilidad



Identificacion del Patron MVC en una Arquitectura de Software de Multiples Capas basada en Componentes

Interfaces
(Pantallas, APIs, etc)

Procesos de Control
(Reglas del Negocio)

Clases de Dominio
(Modelo de Datos)

Mecanismos
(Distribución, Persistencia, etc.)

Servicios de Datos
(RDBMS, LDAP, etc.)



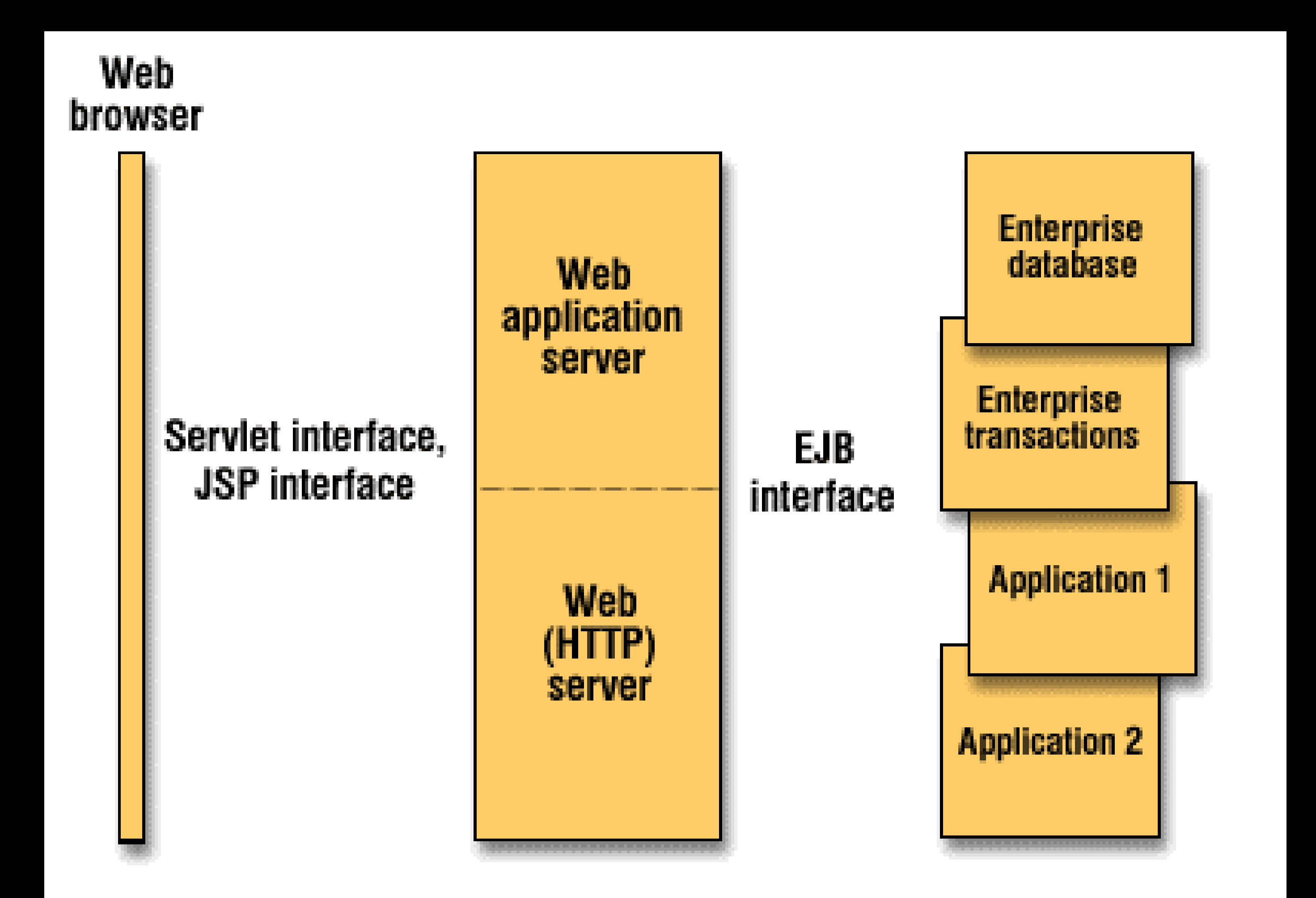
Mecanismos e Infraestructuras de Arquitectura

- Un mecanismo es un componente o grupo de componentes que se utilizan como parte de un patrón para “ayudar” o “dar vida” a los componentes que representan las entidades principales del sistema.
  - Normalmente se ubican en las capas intermedias de la Arquitectura
  - En el patrón MVC, el mecanismo mas común es el que sirve para “hacer persistente” una clase en una base de datos relacional (JDBC, ODBC).

Mecanismos e Infraestructuras de Arquitectura

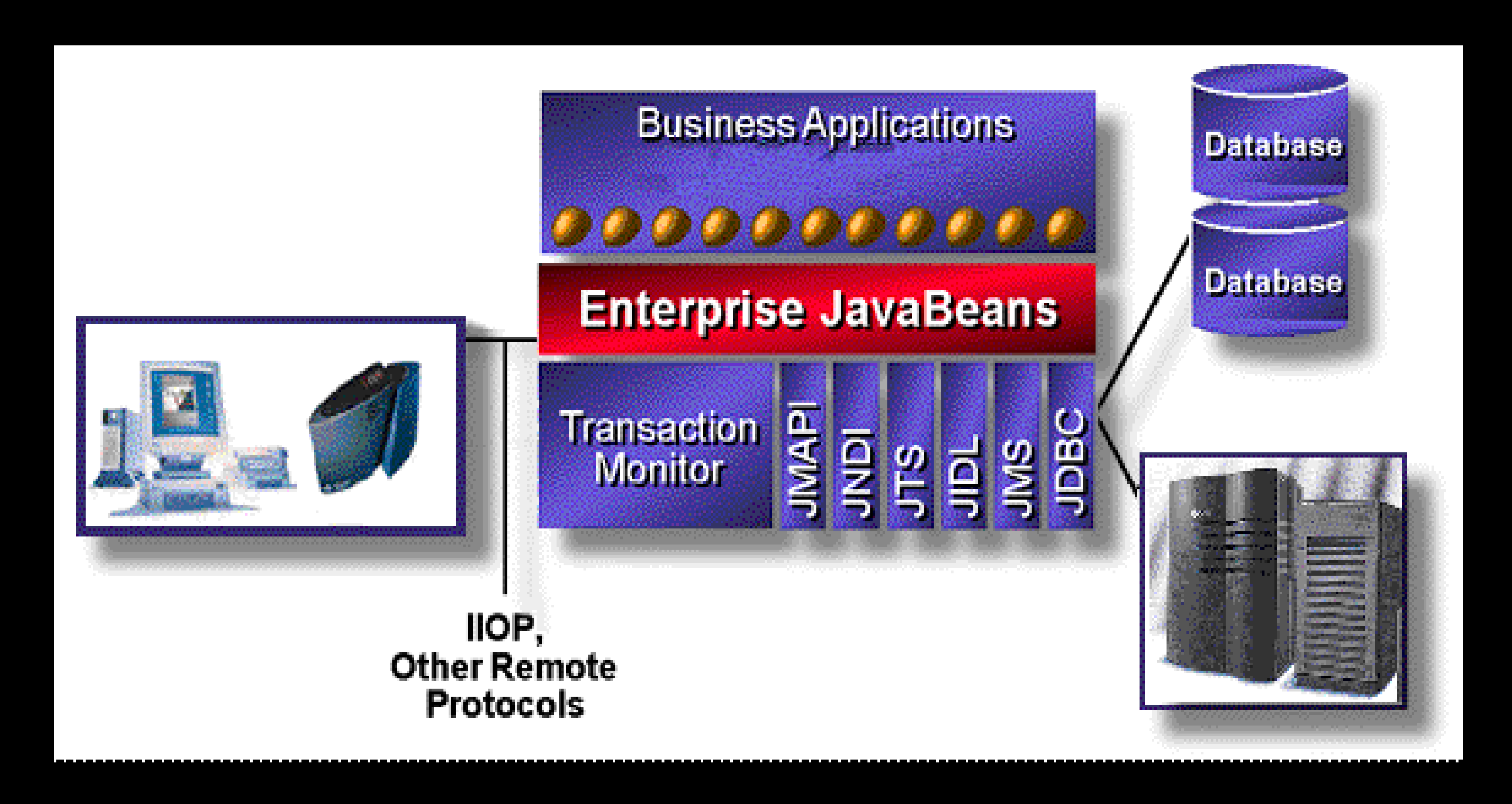
- Una Infraestructura es un grupo de mecanismos que habilitan un conjunto de funcionalidad elemental o servicios básicos (persistencia, transaccionalidad, etc.) y que debe estar disponible globalmente en la arquitectura del sistema.
  - En algunos casos se asigna una capa completa de la arquitectura a una Infraestructura y las capas que se ubican sobre la infraestructura normalmente dependen de ella
  - Las infraestructuras mas populares para implementar software basado en componentes actualmente son el J2EE (EJB, JSP, Servlets, Java Beans, JDBC, Swing, JNDI, JTS, etc.), el DCOM (Distributed Component Object Model) y el CORBA (Common Object Request Broker Architecture)

Mecanismos e Infraestructura de Arquitectura J2EE para Aplicaciones WEB



Componentes J2EE para Ambientes WEB: JSPs, Servlets y EJBs

Mecanismos e Infraestructura de Arquitectura J2EE para Implementar Componentes Transaccionales



Mecanismos e Infraestructura de Arquitectura DCOM

Nodo x

Proxy

Container

In-process server

Nodo y

Standalone server

Container

In-process server



Mecanismos e Infraestructura de Arquitectura

CORBA

Application objects
- Organization specific

CORBA facilities
- User interface
- Information management
- System management
- Task management

CORBA domains
- Financial services
- Health care
- Telecommunications
- Other

Object request broker

CORBA services

- Concurrency	- Lifecycle	- Trade	- Query
- Events	- Naming	- Start up	- Relationships
- Externalization	- Security	- Persistence	- Transactions
- Licensing	- Time	- Properties	- Collections

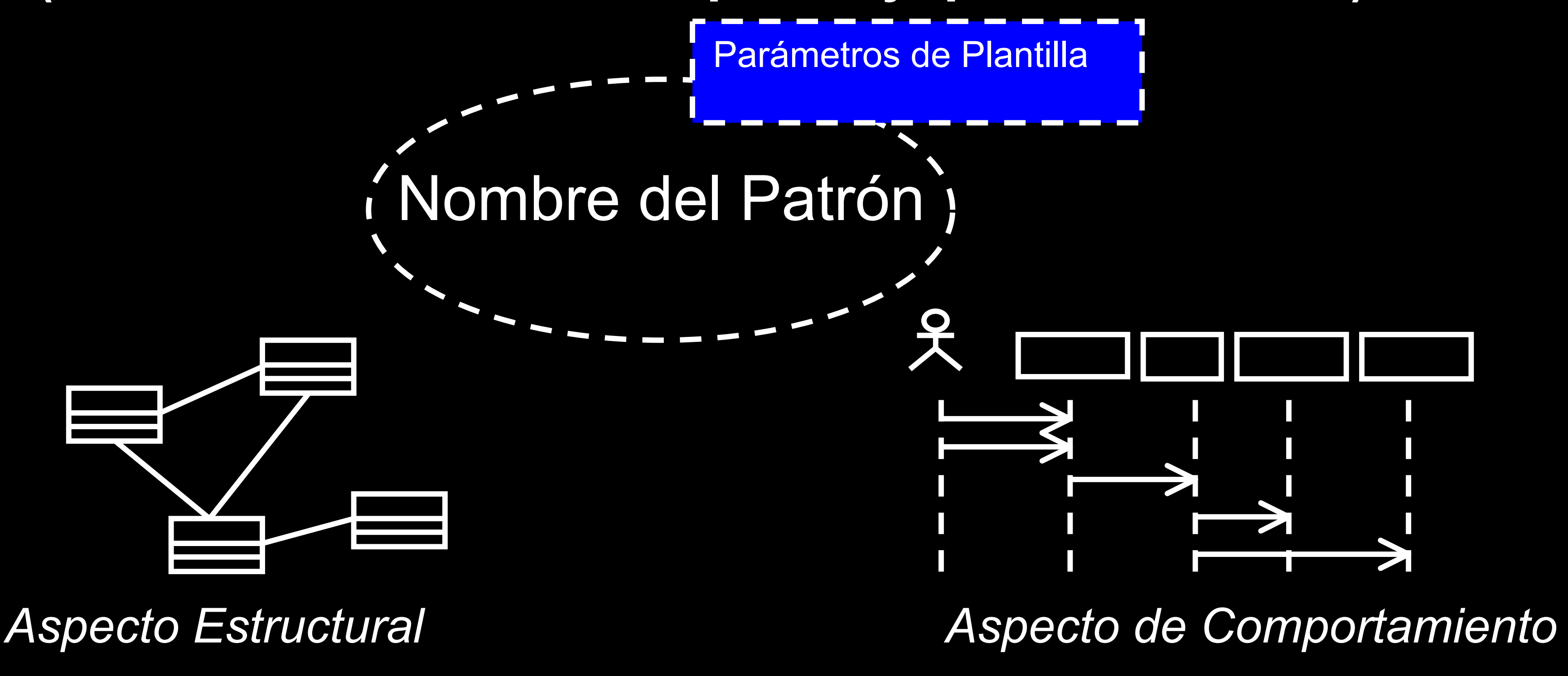
¿Cómo se documenta la Arquitectura de Software?

Documentación de la Arquitectura de Software

- **La arquitectura se documenta en el "Documento de Arquitectura de Software"**
  - Aproximadamente 50 - 60 paginas para un sistema medio; 100 - 120 páginas para un sistema grande
- **El documento incluye:**
  - Una descripción, clasificación y análisis de los requerimientos claves de arquitectura (por ejemplo con el Sistema FURPS+)
  - Una descripción textual del Modelo de Arquitectura
  - Una descripción gráfica (las Vistas) que incluya por lo menos:
    - Escenarios críticos de la arquitectura
    - Una vista de nivel superior de la vista lógica (paquetes y clases clave)
    - Vistas de nivel superior de las vistas de componentes, procesos y producción
  - Guías de Estándares de Diseño y Programación: Contienen los mecanismos de análisis, diseño e implementación
  - Concesiones hechas y alternativas consideradas

Documentando Mecanismos de Arquitectura

- **Para documentarlos gráficamente, los mecanismos de arquitectura se pueden tratar como patrones de diseño (colaboraciones estereotipadas y parametrizadas)**



Documentado en la Guía de Estandares de Diseño

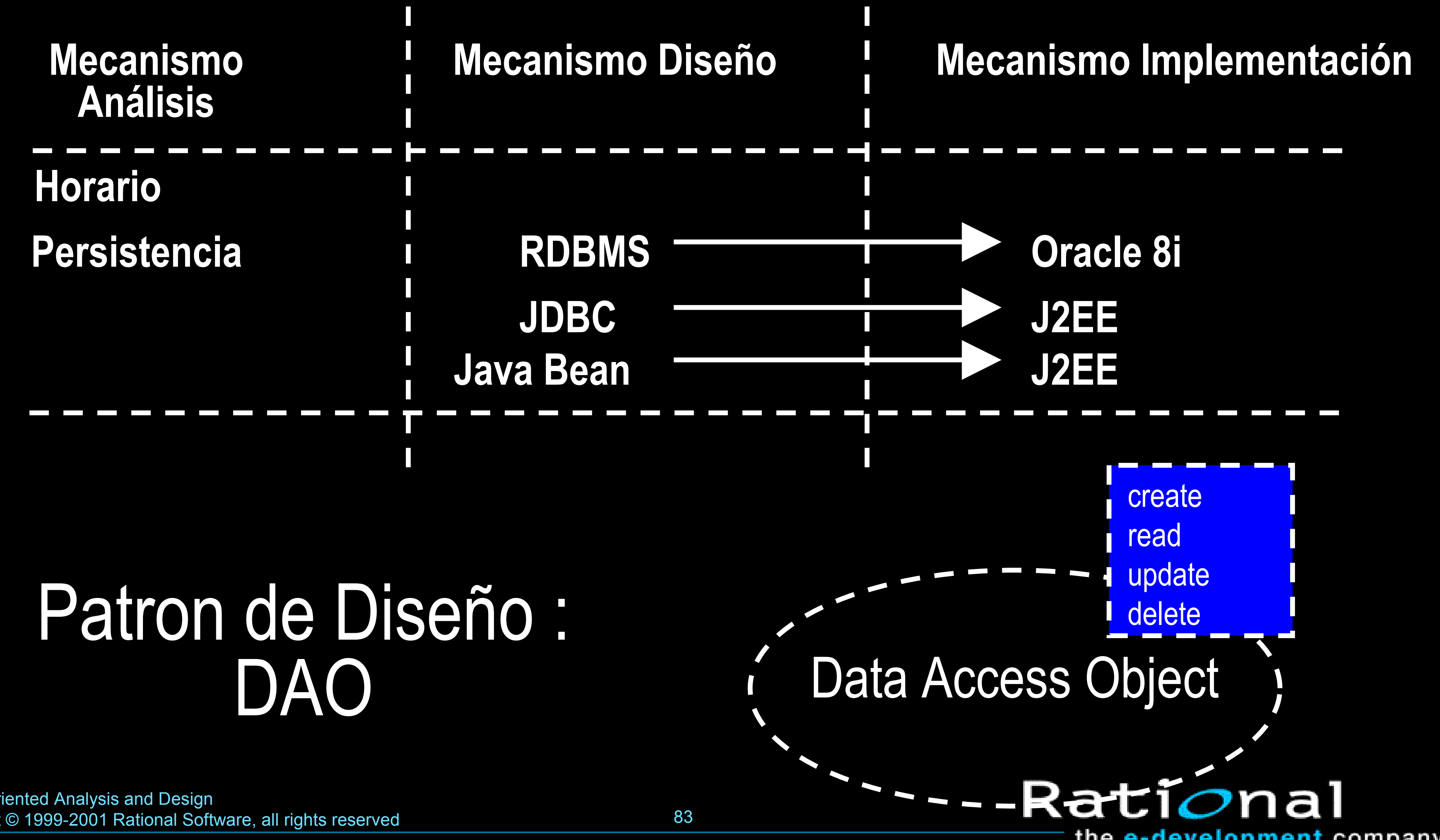
Documentando Mecanismos de Arquitectura

- **Ejemplo: Requerimientos de Persistencia para la clase Horario**
  - Granularidad: 1 to 10 Kbytes por estudiante
  - Volumen: hasta 20,000 horarios por semestre
  - Frecuencia de Acceso
    - Insert: 10000 por día
    - Query: 2000 accesos por hora
    - Update: 2000 por día
    - Delete: 1000 por día

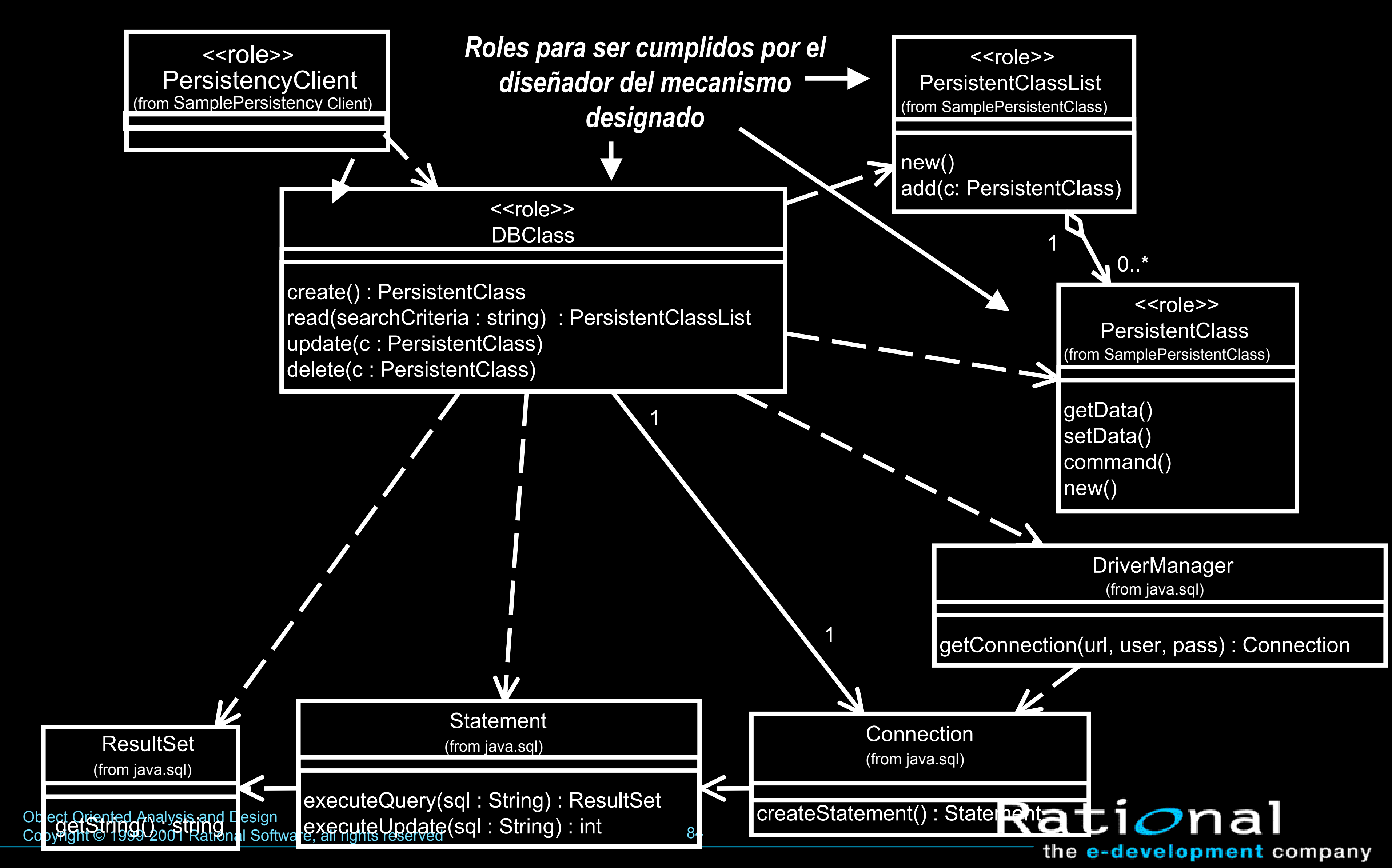


Documentando Mecanismos de Arquitectura

Ejemplo: Persistencia para Horario

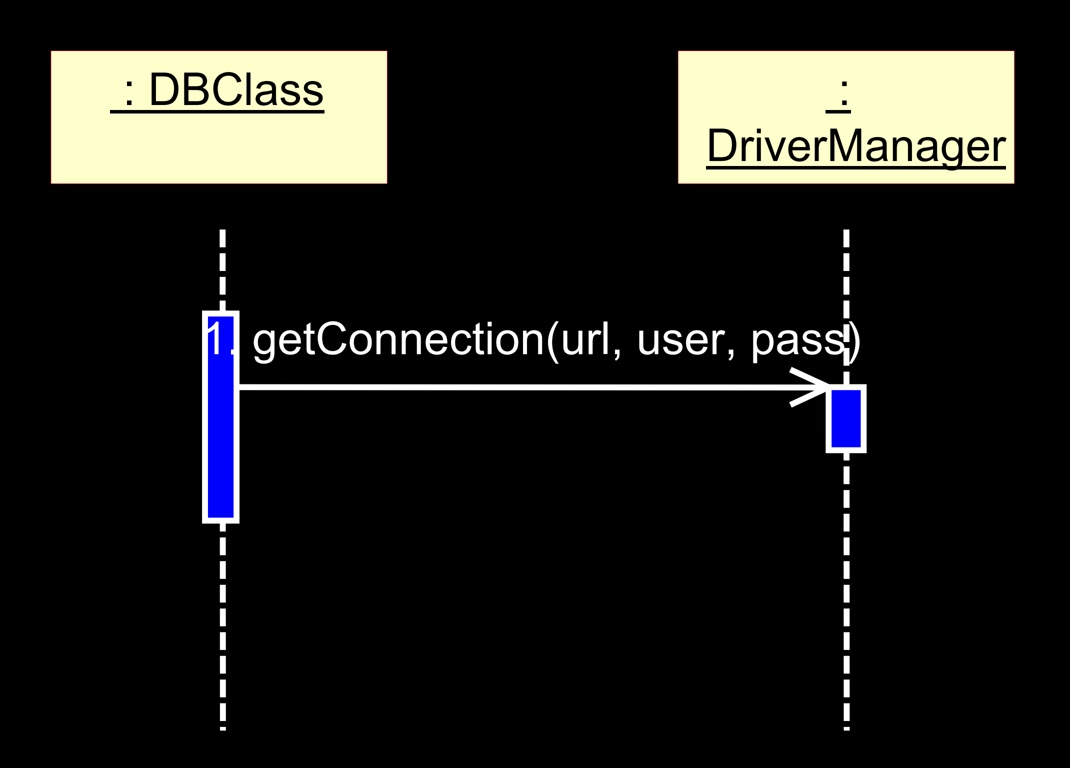


Documentando Mecanismos de Arquitectura



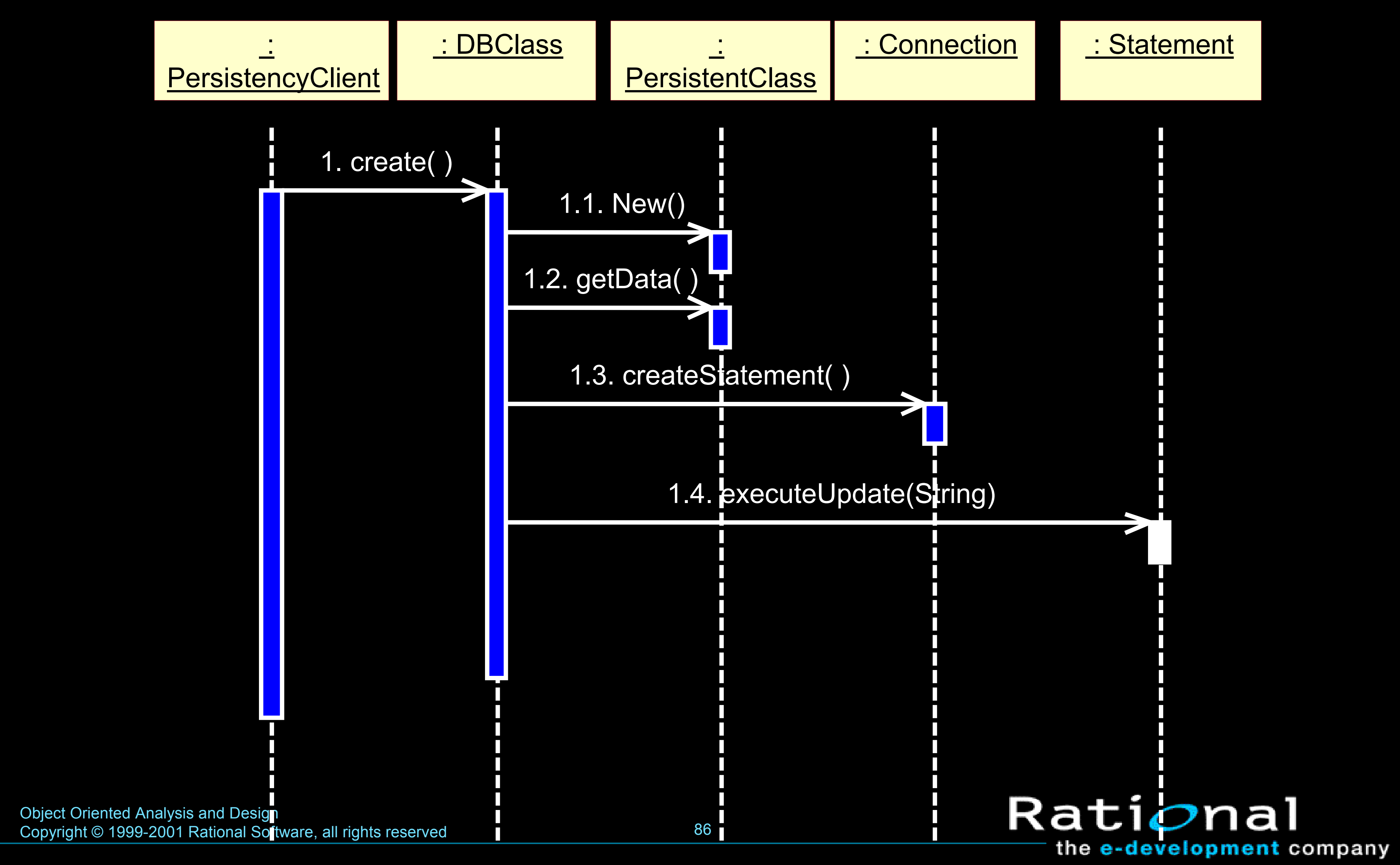
Ejemplo: Persistencia DAO con Java Beans, JDBC y RDBMS

Operación Inicializar



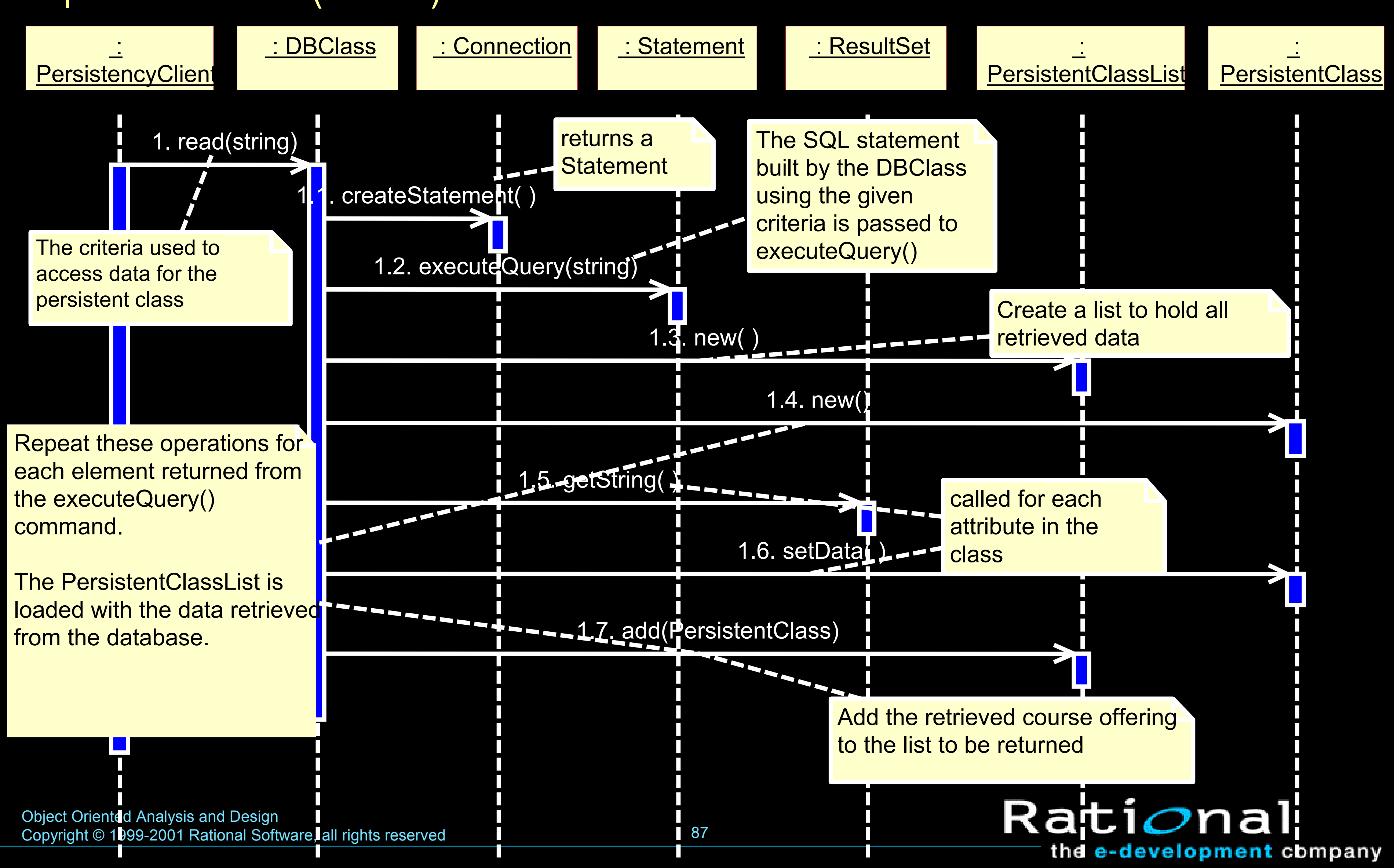
Ejemplo: Persistencia DAO con Java Beans, JDBC y RDBMS

Operación create (Insert)



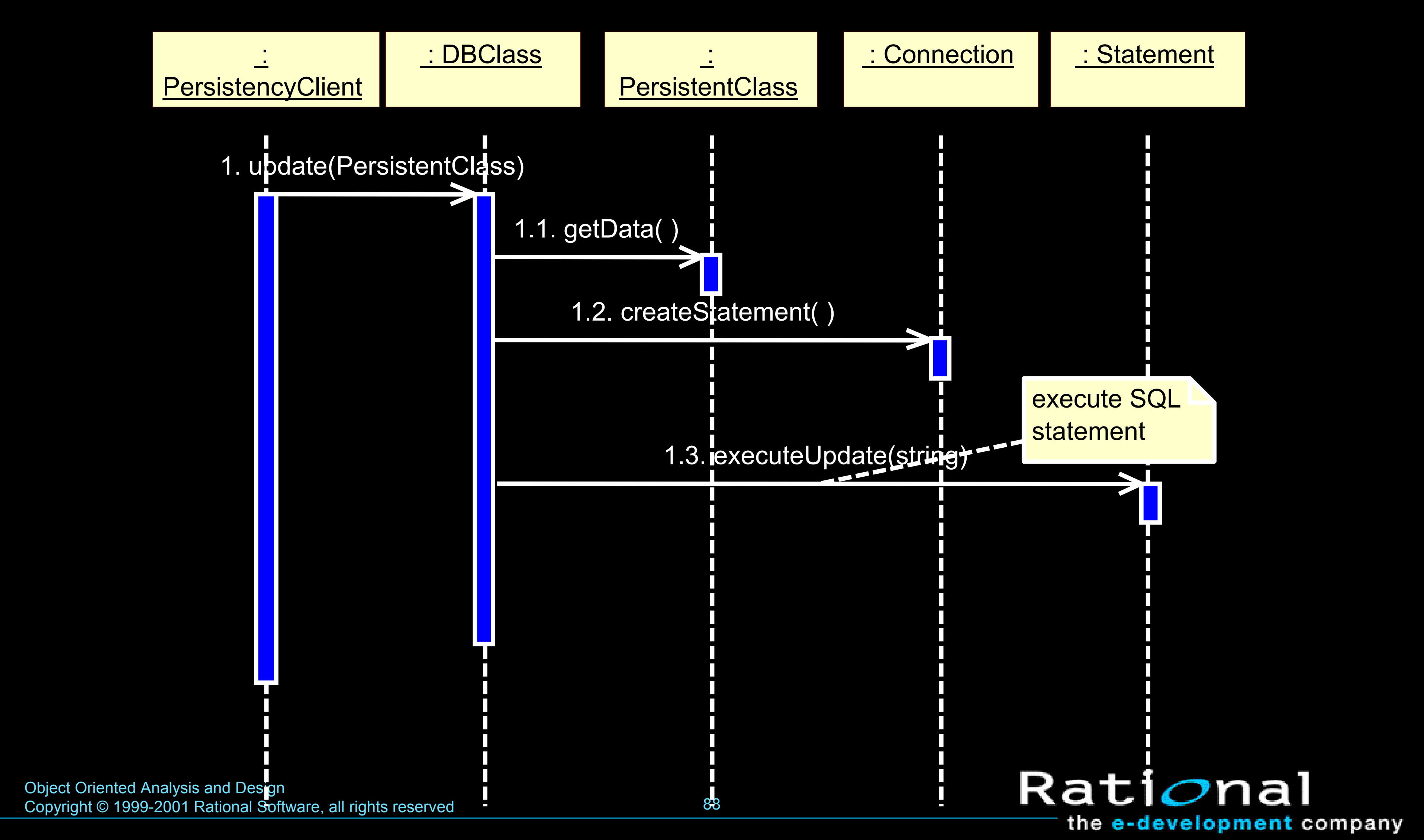
Ejemplo: Persistencia DAO con Java Beans, JDBC y RDBMS

Operación read (Select)



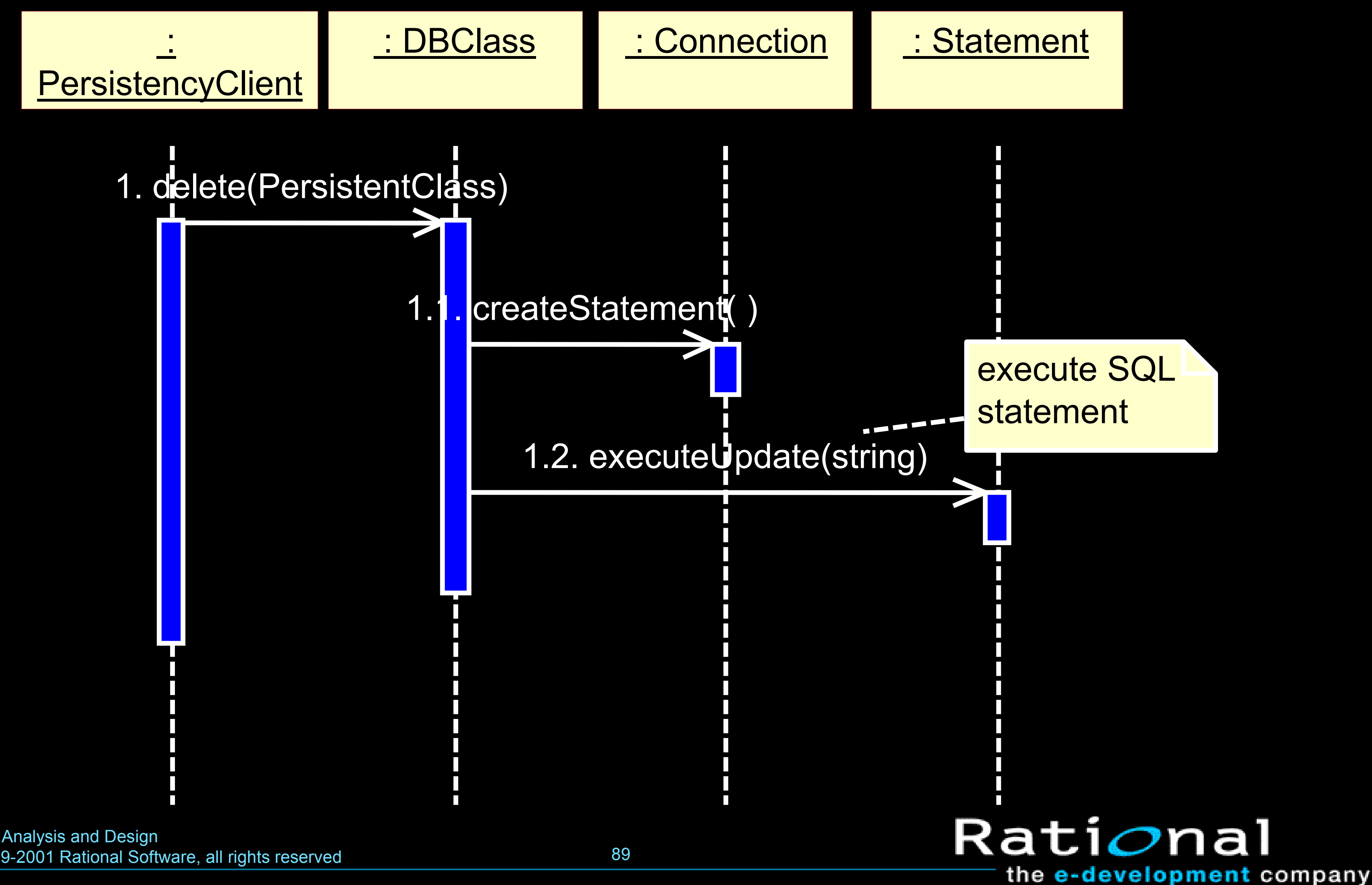
Ejemplo: Persistencia DAO con Java Beans, JDBC y RDBMS

Operación update



Ejemplo: Persistencia DAO con Java Beans, JDBC y RDBMS

Operación delete



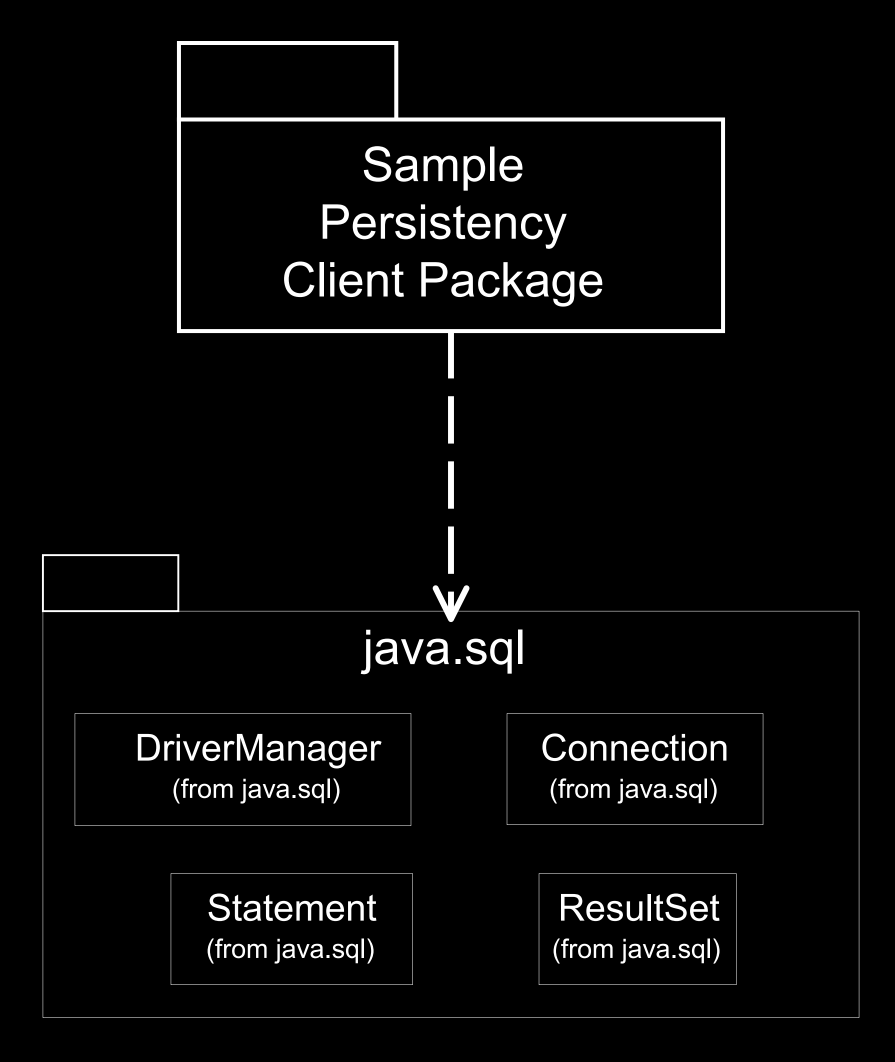
Ejemplo: Persistencia DAO con Java Beans, JDBC y RDBMS

Descripcion Textual de Pasos de Incorporacion

- **Proveer acceso a las librerías de clases necesarias para implementar JDBC**
  - `import java.sql`
- **Crear las necesarias DBClasses**
  - *Una DBClass por cada clase persistente*
- **Incorporar DBClasses en el diseño**
  - Ubicar en paquete/capa
  - Añadir relaciones de los clientes persistentes
- **Crear/Modificar los diagramas de Interacción que describen:**
  - Inicializacion de Base de Datos
  - Acceso a clases persistentes: Create, Read, Update, Delete

Ejemplo: Persistencia DAO con Java Beans, JDBC y RDBMS

Descripcion Grafica de Componentes para Incorporacion



Resumen: Diseño de Arquitectura

- **La Arquitectura de Software decide la organización del software**
- **La Arquitectura de Software maneja los requerimientos del software**
- **Los atributos de buenas arquitecturas son:**
  - Se estructuran en capas con niveles progresivos de abstraccion y dependencia con respecto al sistema que se esta construyendo.
  - En cada capa, se deben utilizar y reutilizar componentes, e infraestructuras de componentes, que han sido estandarizados por la industria
  - Las buenas arquitecturas están basadas en COMPONENTES distribuidos en MULTIPLES CAPAS.

Resumen: Diseño de Arquitectura *(cont.)*

- **La Arquitectura de Software puede definirse según un Modelo de Arquitectura de Software, que incluya:**
  - Patrones de Arquitectura
  - Mecanismos e Infraestructuras de Arquitectura
  - Plataformas Tecnológicas y de Distribución
  - Técnicas de Descripción de la Arquitectura
  - Herramientas o productos utilizados para Implementar la Arquitectura
- **La descripción de la Arquitectura de Software con UML se hace usando el "Modelo de las 4+1 Vistas", que incluye:**
  - Vista de Casos de Uso
  - Vista Lógica
  - Vista de Componentes
  - Vista de Procesos
  - Vista de Producción



Resumen: Diseño de Arquitectura *(cont.)*

- **Para hacer el Diseño de Arquitectura de Software se deben:**
  - establecer y clasificar todos los Requerimientos de Arquitectura
  - definir y documentar los Mecanismos de Arquitectura necesarios para resolver los requerimientos de Arquitectura
  - estructurar los Mecanismos de Arquitectura en múltiples capas según los criterios necesarios para obtener Arquitecturas basadas en componentes
- **El Diseño de Arquitectura debe ser hecho por un Arquitecto o Equipo de Arquitectura según una guía de actividades y responsabilidades establecidas como parte del Proceso Unificado**
- **La documentación de Arquitectura de Software debe incluir:**
  - Una descripción, clasificación y análisis de los Requerimientos de Arquitectura
  - una descripción gráfica de las vistas de la Arquitectura con con diagramas de UML
  - Una descripción, clasificación y análisis de los Mecanismos de Arquitectura

